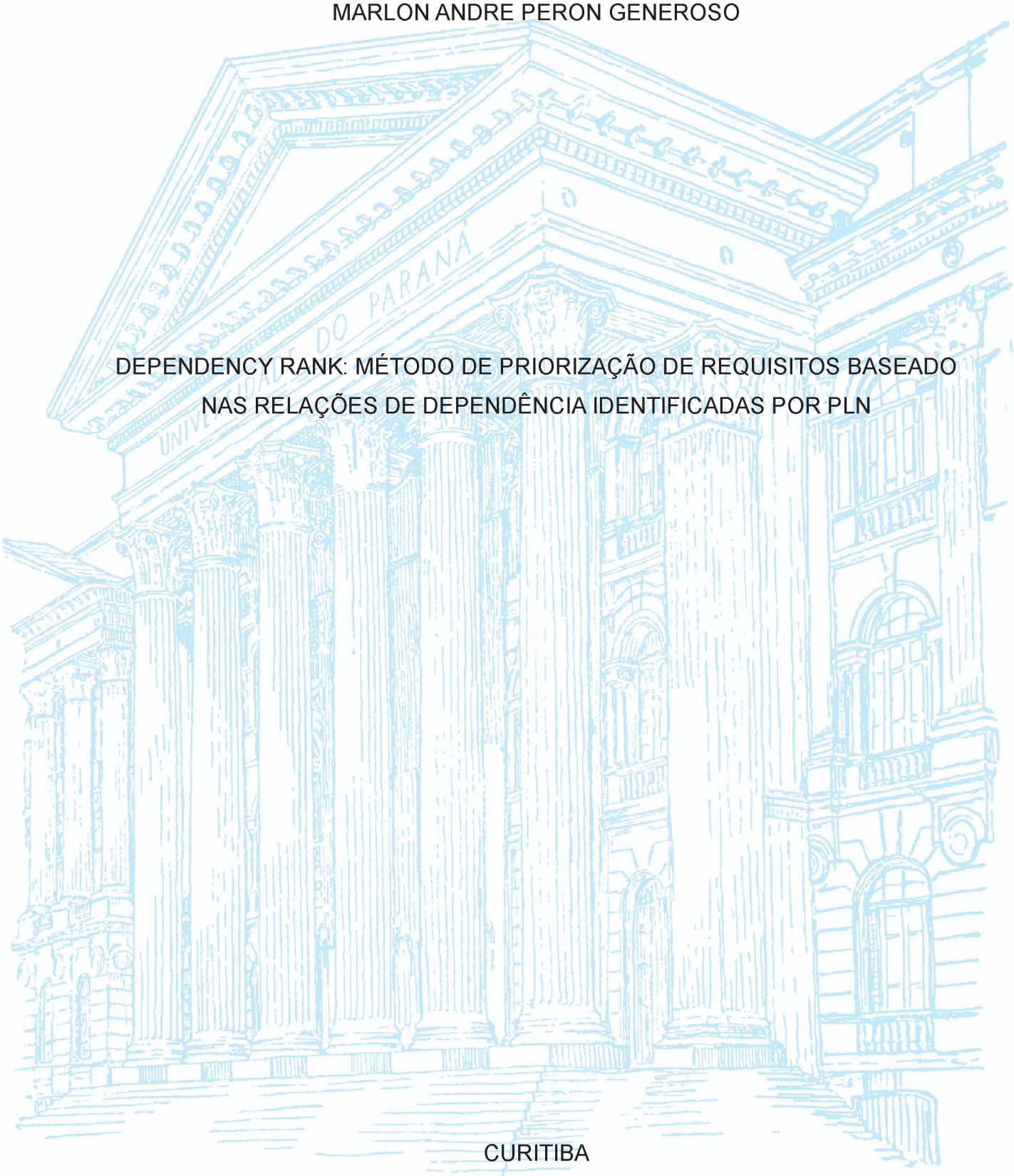
UNIVERSIDADE FEDERAL DO PARANÁ

MARLON ANDRE PERON GENEROSO



DEPENDENCY RANK: MÉTODO DE PRIORIZAÇÃO DE REQUISITOS BASEADO NAS RELAÇÕES DE DEPENDÊNCIA IDENTIFICADAS POR PLN

CURITIBA

2019

MARLON ANDRE PERON GENEROSO

DEPENDENCY RANK: MÉTODO DE PRIORIZAÇÃO DE REQUISITOS BASEADO NAS RELAÇÕES DE DEPENDÊNCIA IDENTIFICADAS POR PLN

Dissertação apresentada ao Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná, como requisito parcial à obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Andrey Ricardo Pimentel

CURITIBA

2019

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

G326d

Generoso, Marlon Andre Peron
Dependency rank: método de priorização de requisitos baseado nas relações de dependência identificadas por PLN [recurso eletrônico] / Marlon Andre Peron Generoso. – Curitiba, 2019.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2019.

Orientador: Andrey Ricardo Pimentel .

1. Processamento de linguagem natural (Computação). 2. Engenharia de software. 3. Software – desenvolvimento. 4. Inteligência artificial. I. Universidade Federal do Paraná. II. Pimentel, Andrey Ricardo. III. Título.

CDD: 001.535

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

MINISTÉRIO DA EDUCAÇÃO
SETOR SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA - 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **MARLON ANDRE PERON GENEROSO** intitulada: **DEPENDENCY RANK: MÉTODO DE PRIORIZAÇÃO DE REQUISITOS BASEADO NAS RELAÇÕES DE DEPENDÊNCIA IDENTIFICADAS POR PLN**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua aprovação no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 25 de Fevereiro de 2019.

ANDREY RICARDO PIMENTEL
Presidente da Banca Examinadora (UFPR)

Natasha Malveira Costa Valentim
NATASHA MALVEIRA COSTA VALENTIM
Avaliador Interno (UFPR)

Maria Cláudia F. P. E.
MARIA CLAUDIA FIGUEIREDO PEREIRA EMER
Avaliador Externo (UTFPR)

Ao meu querido pai José Cid Generoso, que nos deixou há pouco tempo.

AGRADECIMENTOS

Primeiramente agradeço ao meu orientador, o competente e paciente Professor Dr. Andrey Ricardo Pimentel por todos os ensinamentos e incentivos passados durante essa importante etapa da minha vida.

Aos professores do DINF/UFPR que me ajudaram a realizar esse objetivo compartilhando experiências e conhecimentos por todos os semestres.

À minha esposa pela paciência, suporte e companheirismo demonstrado muitas vezes com ajuda da nossa querida Tina.

À minha mãe por me escutar e ceder sua casa sempre que precisei comparecer às aulas ou orientações, tendo deslocamentos mais curtos e, consequentemente, dias mais tranquilos que possibilitaram um aproveitamento melhor.

Aos profissionais que me ajudaram participando dos experimentos deste trabalho, em especial ao coordenador da DSI/CCE/UFPR, Sr. Eduardo Nogueira, que contribuiu com informações importantes para essa fase da pesquisa.

À direção do MAE/UFPR por flexibilizar minha jornada de trabalho possibilitando que eu comparecesse às aulas e orientações em Curitiba.

Ao Prof. Dr. Egon Walter Wildauer, da UFPR, pela carta de recomendação e o incentivo.

Aos demais que sempre me incentivaram e perdoaram os períodos de ausência.

RESUMO

Esta dissertação apresenta um método de priorização de requisitos de *software* baseada em relações de dependência entre funcionalidades. As técnicas de priorização de requisitos de software mais utilizadas atualmente dependem altamente de esforço humano para sua realização, sendo assim, o método proposto buscou diminuir a quantidade de esforço empregada, automatizando parte dessa tarefa, numa tentativa de fornecer maior agilidade e confiabilidade ao processo. Para isso, o método utilizou a documentação de requisitos de um projeto como base para extração dessas relações. Um protótipo que utiliza ferramentas de processamento de linguagem natural foi desenvolvido, sua aplicação teve o objetivo de reconhecer classes candidatas contidas em documentos de especificação de requisitos de *software*, escritos em formato de histórias de usuário, possibilitando, a partir disso, identificar *links* existentes entre as funcionalidades. Após essa análise, um *ranking* sugerido, que emprega como principal critério a priorização dos requisitos com maior número de dependências, é gerado. O método foi testado em dois experimentos, sendo um problema real já implementado e outro hipotético, que teve sua investigação auxiliada por profissionais. Os resultados dos experimentos mostraram que a estratégia implementada para identificação de classes candidatas atingiu, em seu melhor resultado, um F1 score para modelos de classificação de 0,857. Esse índice auxiliou o protótipo a classificar até 70% dos requisitos em intervalos idênticos aos obtidos por julgamento humano, tendo como principal desafio para desenvolvimentos futuros aumentar a carga de subjetividade do método.

Palavras-chave: Priorização de requisitos de *software.* Interdependência entre requisitos. Processamento de linguagem natural. Histórias de usuário. Engenharia de *software*.

ABSTRACT

This dissertation presents a software requirements prioritization method based on dependency relations between features. The most commonly used software requirements prioritization techniques depend heavily on human effort in their performances, so the proposed method intended to reduce the amount of effort employed by automating part of the task in an attempt to improve agility and reliability to the process. Therefore the method used requirements documentations of a software project as a basis for extracting these relations. A prototype that uses natural language processing tools was developed, its application aimed to recognize candidate classes contained in software requirements specification documents, written as user stories, turning possible to identify existing links between the features. After this analysis, a suggested ranking, which employs as the main criterion to prioritize the requirements with greater number of dependencies, is generated. The method was tested in two experiments: a real problem already implemented and another hypothetical, which had its investigation aided by professionals. The results of the experiments showed that the candidate classes identification strategy implemented reached, in its best performance, 0.857 as F1 score for classification models. This index helped the prototype to classify up to 70% of the requirements at the same intervals to those obtained by human judgment. The main challenge for future developments is to increase the subjective analysis of the method.

Keywords: Software requirements prioritization. Requirements interdependency. Natural language processing. User stories. Software engineering.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS OU SIGLAS

AHP	*Analytical Hierarchy Process* (Processo Analítico Hierárquico)
ASCII	*American Standard Code for Information Interchange* (Código Padrão Norte-americano para Intercâmbio de Informações)
ES	Engenharia de *Software*
FRD	*Functional Requirements Dependencies* (Dependências entre requisitos funcionais)
GATE	*General Architecture for Text Engineering* (Arquitetura Geral para Engenharia de Texto)
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto de Engenheiros Eletricistas e Eletrônicos)
PLN	Processamento de linguagem natural
POS	*Part-of-speech* (Parte do discurso)
PRS	Priorização de requisitos de *software*
SIGEOS	Sistema Integrado de Gerenciamento de Estoque e Ordem de Serviço
SLR	*Systematic Literature Review* (Revisão Sistemática da Literatura)
TLR	*Traditional Literature Review* (Revisão Tradicional de Literatura)

UIMA	*Unstructured Information Management Architecture* (Arquitetura de Gerenciamento de Informações Não-estruturada)
UML	*Unified Modeling Language* (Linguagem de Modelagem Unificada)
UTF	*Unicode Transformation Format* (Formato de Transformação Unicode)
XML	*Extensible Markup Language* (Linguagem de Marcação Extensível)
XP	*eXtreme Programming* (Programação Extrema)

SUMÁRIO

1 INTRODUÇÃO

A engenharia de *software* (ES), área da computação que visa estruturar atividades inerentes ao desenvolvimento e manutenção de sistemas de informação é definida por Mall (2018, p. 2, tradução nossa) como uma "coleção sistemática de boas práticas e técnicas de desenvolvimento de programas". A união dessas técnicas possibilita que desenvolvedores profissionais utilizem uma metodologia sistemática de desenvolvimento, facilitando a integração de uma equipe, "caso contrário, eles teriam muita dificuldade em interagir e entender o trabalho do outro e produzir um conjunto coerente de documentos." (MALL, 2018, p. 8, tradução nossa).

Projetos de desenvolvimento de *software* iniciam com a coleta e documentação de requisitos, que são definidos por Bourque e Fairley (2014, p. 1, tradução nossa) como "artefatos que expressam necessidades e restrições inerentes a um produto de *software* que contribui para a solução de um problema do mundo real".

A engenharia de requisitos é a fase da ES em que os requisitos são elicitados. É definida como uma ação que se inicia durante a atividade de comunicação e continua na de modelagem, devendo se adaptar às necessidades do processo, do produto e da equipe (PRESSMAN; MAXIM, 2016, p. 132). Esta ação tem por objetivo ajudar a equipe de desenvolvimento a entender as necessidades do *software* que está sendo proposto, além de especificar de forma correta e completa as funcionalidades desejadas em um projeto.

Em grandes projetos considera-se normal que uma equipe tenha que especificar e gerenciar uma grande quantidade de requisitos, tendo ao mesmo tempo a meta de alcançar a satisfação das partes interessadas (BOURQUE; FAIRLEY, 2014). Diante desse cenário, a adoção de modelos incrementais de desenvolvimento tende a auxiliar uma equipe permitindo avaliar o progresso em direção a essas metas.

Nesses modelos, um incremento é composto de um conjunto de requisitos que formam um produto operacional de *software* para o usuário (PRESSMAN, 2002 citado por PITANGUEIRA *et al.*, 2015). Com o passar dos anos, os modelos incrementais serviram como base para a criação de algumas metodologias ágeis de

desenvolvimento, focadas em diminuir a aparente sobrecarga de trabalho gerada pelos processos tradicionais, como a *eXtreme Programming* (XP).

A metodologia XP utiliza histórias ou cenários para documentar seus requisitos, essas histórias são decompostas em tarefas, priorizadas, estimadas, desenvolvidas e testadas (BOURQUE; FAIRLEY, 2014), gerando, a cada iteração, um incremento de *software*. No entanto, escolher quais histórias do conjunto serão implementadas prioritariamente ou quais eventualmente não serão contempladas em determinado projeto é uma tarefa complexa, com alto grau de subjetividade para uma equipe.

Para Wiegers (1999) a área de estudo sobre Priorização de Requisitos de *Software* (PRS) visa balancear o benefício de determinado requisito para o sistema com seu respectivo custo de desenvolvimento. Várias técnicas de PRS vêm sendo estudadas e testadas no decorrer dos últimos anos, conforme consta em estudo sobre o estado da arte realizado por Achimugu *et al*. (2014). Essas técnicas buscam amenizar o problema da priorização, porém, poucas automatizam o processo de classificação ou parte dele com sucesso.

Nesse contexto, a adoção de linguagem natural para especificação de requisitos em modelos padronizados é frequente, visto que as torna precisas e permite que todos os envolvidos tenham a mesma visão do problema que está sendo estudado, unindo os interesses de clientes e desenvolvedores, formalizando um contrato. Com tal adoção, é possível "fornecer a todas as partes um **entendimento escrito** do problema" (PRESSMAN; MAXIM, 2016, p. 131, *grifo nosso*).

Parte desse entendimento compreende a identificação de *links*, já que sistemas de *software* caracterizam-se pelas conexões entre seus requisitos, então, mesmo que manipulem objetos distintos ou executem operações diversas, estes artefatos tendem a se complementar para atingir algum propósito.

As relações geradas por essas conexões produzem restrições de implementação ou implantação, uma vez que determinada funcionalidade pode condicionar o correto funcionamento de outra, gerando dependências. Portanto, por se tratar de texto livre em linguagem natural, as especificações de requisitos podem revelar essas relações, por exemplo, a partir de citações a objetos manipulados pelas funcionalidades.

Concomitantemente, as soluções de *software* que contemplam Processamento de Linguagem Natural (PLN) para automatização de parte de seu processo estão se tornando cada vez mais frequentes, estando presentes inclusive na *internet* em ferramentas como o Google, com capacidade atual de avaliar milhões de páginas em menos de um segundo com seus algoritmos (CAMBRIA; WHITE, 2014). Assim, pode-se entender que a adoção do PLN tende a simplificar e acelerar processos antes feitos de forma totalmente manual ou guiada, transformando textos em linguagem natural em artefatos compreensíveis computacionalmente.

Em face disso, tem-se como hipótese que o PLN pode auxiliar uma técnica de PRS automatizando algumas etapas a partir da identificação de dependências extraídas por análise textual. Curiosamente, nenhuma das técnicas mapeadas por Achimugu *et al.* (2014) utilizou técnicas de PLN aplicadas em documentos de requisitos em sua abordagem.

Complementarmente, o problema deste trabalho visou responder à seguinte pergunta: É possível priorizar requisitos corretamente identificando dependências através da integração de ferramentas de PLN com uma técnica de PRS?

A principal motivação foi contribuir para a agilidade e a confiabilidade do processo de PRS. Conforme Achimugu *et al.* (2014), em geral, as técnicas propostas atualmente utilizam modelos nos quais uma ferramenta de *software*, que implementa uma estratégia de priorização, recebe de usuários a avaliação individual de prioridade para cada requisito sem qualquer avaliação prévia automatizada, gerando uma lista ordenada de requisitos prioritários baseada apenas no julgamento humano.

Logo, além da demanda adicional por esforço dos profissionais envolvidos, que devem analisar individualmente cada requisito (*e.g.*, com a utilização de técnicas como AHP – *Analytic Hierarchy Process*) (ACHIMUGU *et al.*, 2014); os dados informados por humanos, ainda que com conhecimento e experiência, são baseados em interpretações subjetivas, podendo ser influenciadas por intenções alheias ao foco do problema. Consequentemente, tornou-se desejável prover uma estratégia que possibilitasse amenizar a dificuldade e diminuir o tempo para uma tomada de decisão sobre esse assunto, aumentando a qualidade das avaliações.

1.1 OBJETIVOS

O objetivo geral desta pesquisa foi desenvolver um método de priorização de requisitos de *software* baseado nas relações de dependência extraídas a partir de especificações feitas em linguagem natural.

Para tanto, os seguintes objetivos específicos foram contemplados:

a) Realizar estudo sobre o estado da arte em PRS, analisando o tratamento das relações de dependência nas abordagens;
b) Desenvolver algoritmo para identificação de classes candidatas de um sistema analisando especificações em linguagem natural;
c) Identificar automaticamente as dependências entre requisitos por meio das referências feitas às classes candidatas reconhecidas;
d) Desenvolver algoritmo de PRS que priorize automaticamente os requisitos com maior número de dependências.
e) Comparar os resultados obtidos pelo protótipo desenvolvido com análises de profissionais da área, avaliando o índice de correção atingido e possíveis limitações.

O estudo sobre o estado da arte em PRS forneceu um panorama da área de pesquisa evidenciando como as relações de dependência entre requisitos são tratadas pelas técnicas/métodos mais utilizados. De posse desses dados foi possível planejar a arquitetura do método proposto considerando as limitações tecnológicas do escopo.

A identificação de classes candidatas a partir de linguagem natural consistiu em extrair das especificações de requisitos os termos chave para o contexto da aplicação, interpretando-os como possíveis objetos do sistema em questão. A correção na identificação dessas classes/objetos foi necessária para a análise de dependências entre os requisitos.

As dependências foram definidas a partir da referência de dois ou mais requisitos a um mesmo objeto, pois, considerou-se alta a probabilidade de haver relação de restrição ou similaridade entre artefatos com essa característica.

O critério implementado priorizou os requisitos com maior número de dependências encontradas. Esse critério foi definido por auxiliar a responder a questão de pesquisa e não depender de julgamentos adicionais.

A comparação com dados obtidos por profissionais auxiliou na validação dos resultados de forma semelhante à feita sem a adoção de uma ferramenta de *software*, possibilitando identificar, por exemplo, em quais pontos o método precisa ser melhorado.

1.2 METODOLOGIA

A realização de experimentos foi de grande importância para este trabalho, pois possibilitou a comparação dos resultados auferidos manualmente pelos desenvolvedores com os do método proposto. A aplicação a um problema real também demandou a realização de experimentos, avaliando um sistema já priorizado por uma equipe de desenvolvedores. Dessa forma, viabilizou-se a validação das hipóteses levantadas, a identificação de limitações e a aferição de proximidade do método.

Diante disso, esta pesquisa focou em um delineamento quase experimental conforme definido por Batista e Campos (2007), pois implementou controles para que variáveis estranhas ou intervenientes não comprometessem os resultados do experimento, além de não contar com randomização da amostra.

Complementarmente, foram realizadas pesquisas bibliográficas dos conceitos envolvidos no escopo da pesquisa, nas áreas de ES e PLN, sendo: requisitos, desenvolvimento ágil de *software*, interdependência entre requisitos, PRS, preprocessamento de texto, *part-of-speech* (POS) *tagging* e lematização.

Os principais métodos de PRS foram investigados em um mapeamento sistemático de literatura. De acordo com Petersen *et al.* (2008), esses estudos têm o propósito de fornecer uma visão geral de uma área de pesquisa por meio da classificação e contagem de contribuições. Publicações cujas abordagens contemplaram a extração de requisitos ou recursos de *software* auxiliada por PLN também foram investigadas e, assim como o mapeamento citado, compõem o capítulo sobre trabalhos correlatos.

A pesquisa utilizou a técnica de métodos mistos, complementando resultados obtidos com métodos qualitativos e quantitativos (CRESWELL, 2007), com a utilização de questionário disponibilizado na forma de uma página na *internet* como técnica de coleta de dados. O questionário contou com perguntas abertas e

fechadas que se referiam tanto aos dados de entrada do método quanto aos resultados obtidos como saída pelos profissionais.

Por fim, quanto ao objeto de estudo, tratou-se de amostragem não probabilística por julgamento. Segundo Appolinário (2012, p. 135), esse tipo de amostragem é utilizado quando deseja-se fazer uma pesquisa avaliando a opinião de especialistas em determinada área. Assim, os sujeitos são escolhidos intencionalmente pelo pesquisador. Nesse caso, foi estabelecido como critério de seleção a formação superior na área de tecnologia da informação e conhecimento em PRS.

1.3 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado de acordo com a seguinte sequência de capítulos:

Capítulo 1: compreende a introdução do trabalho, o contexto da pesquisa, seus objetivos, hipóteses, motivações e a metodologia utilizada.

Capítulo 2: apresenta a revisão de literatura dos conceitos envolvidos, aplicados ao método proposto.

Capítulo 3: exibe uma visão geral sobre os trabalhos correlatos que embasaram o desenvolvimento desse estudo em um mapeamento sistemático da literatura em PRS, citando características, limitações e semelhanças.

Capítulo 4: apresenta a estrutura do método proposto comentando cada uma de suas camadas. Também é apresentado o protótipo desenvolvido com as principais rotinas implementadas.

Capítulo 5: demonstra detalhadamente os experimentos realizados, seus valores de entrada, sujeitos e resultados obtidos.

Capítulo 6: Apresenta as considerações finais do trabalho, cita as principais colaborações, as limitações encontradas, possíveis focos para futuras pesquisas e a conclusão do autor.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta os principais conceitos utilizados durante o desenvolvimento do presente trabalho, visando o entendimento do processo de PRS, das técnicas de PLN utilizadas e demais conceitos relacionados.

2.1 REQUISITOS DE *SOFTWARE*

Requisitos de *software* podem ser definidos como uma propriedade que deve ser exibida por algo para resolver um problema no mundo real (BOURQUE; FAIRLEY, 2014). Eles expressam necessidades ou restrições desejadas em um contexto onde um *software* é aplicável e surgem de situações variadas, já que, de acordo com Pressman e Maxim (2016, p. 132), projetos de desenvolvimento de *software* tendem a começar inclusive durante conversas informais, quando se percebe a necessidade de desenvolver uma ferramenta para atender a um conjunto de necessidades e características ainda não disponível.

Bourque e Fairley (2014) complementam que, somado ao propósito de automatizar parte de uma tarefa para alguém apoiando os processos de negócios de uma organização, requisitos também servem para representar demandas de correção de deficiências de um *software* já existente ou de controle de dispositivos, assim, novos dispositivos e *softwares* preexistentes também podem estabelecer situações em que requisitos sejam identificados.

Na ES os requisitos podem ser divididos entre funcionais e não-funcionais. Requisitos funcionais "representam os comportamentos que um programa ou sistema deve apresentar diante de certas ações de seus usuários" (PAULA FILHO, 2003, p. 13). De acordo com Zhang (2013), um conjunto de requisitos funcionais relacionados logicamente compõem um recurso, que fornece uma capacidade ao usuário permitindo a satisfação de um objetivo do negócio. Ou seja, requisitos funcionais referem-se às funcionalidades desejadas para determinado sistema em estudo.

Por sua vez, os requisitos não-funcionais são definidos por Capilla (2012 citado por Misaghian, 2018) como a atuação de um sistema em sua funcionalidade em termos de atributos de qualidade como confiabilidade, segurança, desempenho ou usabilidade. Para Paula Filho (2003, p. 13), esses atributos "quantificam

determinados aspectos do comportamento" desses sistemas, sendo classificados como requisitos não-funcionais por não representarem diretamente uma funcionalidade ou capacidade desejada pelo usuário.

A engenharia de requisitos é a fase do processo de *software* em que os requisitos são minuciosamente estudados. É definida como uma ação que se inicia durante a atividade de comunicação e continua na de modelagem, devendo se adaptar às necessidades do processo, do produto e da equipe (PRESSMAN; MAXIM, 2016, p. 132), tendo por finalidade auxiliar a equipe no entendimento do *software* proposto e estabelecer comunicação eficaz com os clientes.

Paula Filho (2003, p. 14) destaca a importância da fase de comunicação nessa etapa, sugerindo que "a participação dos usuários na engenharia de requisitos é fundamental para que as necessidades deles sejam corretamente atendidas pelo produto".

Após coletados, os requisitos passam pela etapa de análise, que, segundo Bourque e Fairley (2014) visa detectar e resolver conflitos entre eles. Os documentos de especificação são comumente usados pelas equipes para fomentar essa análise, pois permitem a identificação desses conflitos. Essa identificação é possível a partir de análise textual das especificações, já que a linguagem natural "normalmente é usada para expressá-los" (PAULA FILHO, 2003, p. 14), tornando-as precisas e de fácil entendimento a todos os envolvidos.

Este trabalho foca em documentos de especificação de requisitos funcionais, uma vez que são esses documentos que armazenam as informações sobre classes/objetos manipulados e podem revelar relações de dependência entre as funcionalidades. É possível perceber que esses documentos se inserem no processo de *software* como artefatos importantes também para a comunicação entre os interessados e a documentação do sistema.

A adoção de linguagem natural em sua elaboração tem papel fundamental na comunicação possibilitando que todos os envolvidos tenham um entendimento parecido sobre o que está sendo planejado. Tal adoção pode ajudar a revelar informações importantes também em processos de priorização, já que as relações de dependência entre requisitos, inerentes ao princípio da coesão de um sistema de informação podem estar implícitas nessas descrições, conforme exposto adiante.

2.2 DESENVOLVIMENTO ÁGIL DE *SOFTWARE*

Para Da Silva e Videira (2001, p. 31) um processo de *software* "pretende designar uma sequência de atividades, normalmente agrupadas em fases e tarefas, que são executadas de forma sistemática e uniformizada" para o desenvolvimento de um produto de *software*. Esse processo pode ser dividido em 3 fases principais, sendo concepção, implementação e manutenção. A fase de concepção se preocupa em identificar **o que** o sistema deve fazer. A implementação foca em **como** o sistema será feito, enquanto a manutenção é a fase em que são feitas as alterações necessárias após a aceitação do produto (DA SILVA; VIDEIRA, 2001, p. 41-42).

A partir da década de 1990, com o objetivo de diminuir a aparente sobrecarga de trabalho gerada pelos processos de *software* tradicionais, os chamados métodos ágeis de desenvolvimento começaram a ser elaborados, possuindo como características ciclos curtos e iterativos, equipes auto-organizáveis, frequente envolvimento dos clientes e ênfase na criação de produtos demonstráveis a cada ciclo (BOURQUE; FAIRLEY, 2014), ou seja, essas metodologias caracterizam-se também por produzir *software* de maneira incremental, já que enfatizam a necessidade de criar sucessivamente versões demonstráveis do produto.

No ano de 2001, 16 renomados profissionais e estudiosos assinaram o Manifesto Ágil, que declara valorizar "*software* em funcionamento mais que documentação abrangente" e "colaboração com o cliente mais que negociação de contratos" (Manifesto para o desenvolvimento ágil de *software*, 2001), entre outros princípios que diferenciavam essas metodologias dos processos tradicionais. O manifesto unido à definição de "escolher o conjunto apropriado de ações" apresentado anteriormente, embasou a criação das metodologias ágeis mais difundidas atualmente e auxiliou na definição de atividades auxiliares utilizadas por elas.

Nesse contexto, as metodologias ágeis geralmente definem rotinas com o intuito de facilitar a adaptação de mudanças no projeto, a comunicação da equipe (evitando as várias instâncias hierárquicas) e, diminuindo a importância dos artefatos intermediários (documentação extensa) tornando possível focar, de forma incremental, em um produto mais completo em funcionamento a cada ciclo.

Para exemplificar a aplicação dos princípios declarados no manifesto ágil, uma das metodologias mais difundidas na atualidade[1], a XP (*eXtreme Programming*) contempla a elaboração de testes a cada incremento, possui envolvimento direto do cliente com a equipe e utiliza histórias ou cenários para especificar requisitos (BOURQUE; FAIRLEY, 2014), permitindo inclusive que o cliente indique suas prioridades em relação a eles. Logo, ainda que as decisões de planejamento do projeto estejam sob o controle da equipe de desenvolvimento, a participação do cliente é fundamental no auxílio à tomada dessas decisões.

Conforme exposto, as metodologias ágeis visam simplificar o processo de desenvolvimento focando em atividades essenciais como a implementação, elaborando apenas a documentação necessária, contando sempre com a cooperação com o cliente.

O modelo de especificação de requisitos sugerido pela metodologia XP, chamado de histórias de usuário, corrobora com o princípio do envolvimento do cliente no processo e é detalhado a seguir.

2.2.1 Histórias de Usuários (*User Stories*)

As histórias de usuário representam uma técnica de especificação de requisitos comumente usada em metodologias ágeis de desenvolvimento de *software*, de acordo com Bourque e Fairley (2014), tratam-se de descrições curtas e de alto nível da funcionalidade exigida, expressas em termos do cliente. Cada história "descreve o resultado, as características e a funcionalidade solicitada para o *software* a ser construído" (PRESSMAN; MAXIM, 2016, p. 73) utilizando linguagem natural. Portanto, seu formato simples de descrição vai ao encontro do princípio ágil da colaboração com o cliente, definindo em termos claros quem deseja, o que é desejado, e para que a função é importante no contexto do *software*.

Bourque e Fairley (2014) sugerem que uma história de usuário deve conter apenas informações suficientes para que os desenvolvedores possam produzir uma estimativa razoável do esforço para implementá-la, para tal, esse modelo simplificado de especificação geralmente possui o formato apresentado a seguir:

As a <role>, I want <goal/desire> so that <benefit>.

(BOURQUE; FAIRLEY, 2014).

---

1 Segundo Pressman e Maxim (2016, p. 72).

Na sentença, *role* (papel) se refere ao papel desempenhado por quem deseja a funcionalidade, ou seja, qual tipo de usuário demanda o requisito. *Goal/Desire* (objetivo/desejo) define de forma direta qual funcionalidade é desejada pelo *role* para o sistema. Por fim, *benefit* (benefício) apresenta o possível ganho alcançado com o desenvolvimento da função solicitada.

Além de documentar requisitos, especificando-os de acordo com a visão do cliente, outra possibilidade desses documentos é auxiliar na priorização de requisitos, situação também contemplada pela metodologia XP, por exemplo. Segundo Pressman e Maxim (2016, p. 73),

> Cada história é escrita pelo cliente e é colocada em uma ficha. O cliente atribui um valor (uma prioridade) à história baseando-se no valor do negócio global do recurso ou função. Os membros da equipe XP avaliam, então, cada história e atribuem um custo – medido em semanas de desenvolvimento – a ela. [...] Clientes e desenvolvedores trabalham juntos para decidir como agrupar histórias para a versão seguinte.

A documentação de requisitos utilizando histórias de usuário facilita a participação do cliente e o entendimento de todos os envolvidos. Como pôde ser observado, a adoção de linguagem natural em sua composição torna a definição clara, aproximando a percepção de necessidades, papéis e benefícios entre os envolvidos no projeto.

Por se tratar de texto livre, porém parametrizado, os campos "*role*", "*goal/desire*" e "*benefit*" tendem a armazenar dados relevantes sobre como os requisitos se complementam para atingir o objetivo geral de um sistema e em qual ambiente serão executados, ou seja, o conteúdo desses campos auxilia na definição do agrupamento de histórias para a próxima versão, também chamado de incremento.

Para o contexto deste trabalho, conjuntos de histórias de usuário são submetidos como entradas de um método de PRS, buscando por meio da análise textual desses documentos, auxiliada por PLN, encontrar evidências de *links* de dependência entre os requisitos do problema. Sendo assim, o método proposto não utiliza os custos atribuídos às histórias em sua análise, diferenciando-o do processo executado pela metodologia XP. Os conceitos relacionados a interdependência, PRS e PLN são detalhados nas próximas seções.

2.3 INTERDEPENDÊNCIAS ENTRE REQUISITOS

Um conjunto de requisitos deve possuir dependências intrínsecas entre eles, caso contrário nunca poderá descrever um sistema com coesão suficiente (ZHANG, 2013). A coesão entre requisitos deriva da característica desses artefatos se complementarem para atingir os objetivos de um sistema de informação, gerando *links* entre as funcionalidades desejadas pelo cliente. Esses *links* são amplamente abordados em publicações na área de requisitos de *software* cujo foco é a rastreabilidade.

Em um dos primeiros estudos com grande relevância na área, Pohl (1996 citado por Dahlstedt e Persson, 2003) desenvolveu um *framework* para rastreabilidade que incluiu dezoito possíveis tipos de *links* de dependência que poderiam ser aplicados a qualquer tipo de objeto rastreável usado no processo de engenharia de requisitos, porém, nem todos esses tipos podem ser aplicados a requisitos de *software*.

Após realização do mapeamento sistemático da literatura apresentado no Capítulo 3, identificou-se que o tratamento de dependências implementado pelos estudos selecionados levou em conta 3 possíveis classificações para essas relações. As classificações são apresentadas a seguir, tendo suas estruturas exibidas nos Quadros 1, 2 e 3.

Carlshamre *et al.* (2001) elaboraram um estudo aprofundado sobre interdependências a partir de cinco conjuntos distintos, baseou-se em entrevistas com engenheiros de requisitos e nos resultados de seu experimento para propor a classificação exibida no Quadro 1.

QUADRO 1: CLASSIFICAÇÃO SEGUNDO CARLSHAMRE *ET AL.* (2001)

Tipo	Significado
R_1 *AND* R_2	R_1 requer R_2 para funcionar e R_2 requer R_1 para funcionar.
R_1 *REQUIRES* R_2	R_1 requer R_2 para funcionar mas o contrário não é verdadeiro.
R_1 *CVALUE* R_2	R_1 afeta positiva ou negativamente o valor de R_2 para o cliente.
R_1 *ICOST* R_2	R_1 afeta positiva ou negativamente o custo de implementação de R_2.

FONTE: Adaptado de Carlshamre *et al.* (2001).

É possível perceber que os *links* de dependência afetam o planejamento de um *release*[2] de maneiras diferentes. As relações *CVALUE* e *ICOST* são baseadas

2 Versões de um *software*, denominados incrementos, que fornecem progressivamente mais funcionalidade para o cliente à medida que cada um é entregue (PRESSMAN, 2010, p. 41).

em valores, indicam que a seleção de determinado requisito pode impactar na percepção do valor ao cliente ou no custo de implementação de outros requisitos, nesses casos, a identificação possui maior carga de subjetividade sendo uma tarefa prioritariamente realizada por profissionais experientes, conforme pôde ser observado nos estudos citados.

As relações *AND* e *REQUIRES* focam em questões funcionais, indicando que existem possíveis preceitos que devem ser contemplados. Por exemplo, em um sistema para um restaurante que possua entre seus requisitos "R_1: cadastrar itens do menu" e "R_2: realizar pedido de itens do menu", pode-se deduzir que existe uma relação do tipo R_2 *REQUIRES* R_1, pela necessidade de possuir itens cadastrados antes de realizar um pedido.

De forma adicional a Carlshamre *et al.* (2001), Dahlstedt e Persson (2003) aprofundaram o assunto definindo novas categorias para a classificação de interdependências separando-as em "*structural*" (estruturais) e "*cost/value*" (de custo/valor). As categorias propostas por Dahlstedt e Persson (2003) são exibidas no Quadro 2.

QUADRO 2: CLASSIFICAÇÃO SEGUNDO DAHLSTEDT E PERSSON (2003).

Categoria	Tipo	Significado
Structural	*Requires*	Dependência hierárquica, similar a R_1 *REQUIRES* R_2.
	Explains	Um requisito mais genérico pode ser explicado por um conjunto de requisitos mais específicos.
	Similar_to	Indica similaridade entre os requisitos relacionados.
	Conflicts_with	Os requisitos não devem ocorrer ao mesmo tempo.
	Influences	Influência estrutural de forma diferente das anteriores.
Cost/Value	*Increases/Decreases _cost_of*	Influência no custo de implementação de outro requisito, similar a R_1 *ICOST* R_2.
	Increases/Decreases _value_of	Influência na percepção do valor ao cliente, similar a R_1 *CVALUE* R_2.

FONTE: Adaptado de Dahlstedt e Persson (2003).

O maior nível de detalhes da abordagem de Dahlstedt e Persson (2003) fica evidente na categoria de dependências estruturais já que as relações *AND*, *REQUIRES* e *OR*[3] propostas por Carlshamre *et al.* (2001) não fornecem suporte ao mapeamento de requisitos complementares (*Explains*), similares (*Similar_to*) ou com

3 Apesar de não constar no Quadro 1, a relação *OR* foi investigada por Carlshamre *et al.* (2001), constando em seu conjunto preliminar de interdependências. Possui características equivalentes a *Conflicts_with* de Dahlstedt e Persson (2003).

outros tipos de influência não parametrizada (*Influences*) propostos por Dahlstedt e Persson (2003).

Por fim, Zhang (2013) elaborou um modelo de dependências focando apenas em requisitos funcionais (FRD – *Functional Requirements Dependencies*). Nesse modelo, conforme mostra o Quadro 3, as dependências relacionadas a custo/valor não foram contempladas por possuírem maior efeito em requisitos não-funcionais. Por suposição, foi considerado que as dependências do tipo "*Similar_to*", apresentadas em Dahlstedt e Persson (2003) não seriam relevantes por serem resolvidas na fase de especificação funcional (ZHANG, 2013, p. 22).

QUADRO 3: CLASSIFICAÇÃO SEGUNDO ZHANG (2013).

Categoria	Tipo	Significado
Structural	*Elaboration*	Se refere às dependências entre requisitos que compõem uma funcionalidade em comum, ou seja, é o conjunto obtido a partir da decomposição da funcionalidade em requisitos.
Constraint	*Requiring*	Similar a R_1 *REQUIRES* R_2.
	Exclusion	Similar a *Conflicts_with*.
Operational	*Sequence*	Indica que a ação de R_1 ocorre logo após a ação de R_2 ser completada.
	Parallelism	Indica que dois ou mais requisitos funcionais podem ocorrer paralelamente.

FONTE: Adaptado de Zhang (2013, p. 24-27)

Diante do exposto, é possível notar que ao longo dos últimos vinte e dois anos foram obtidas importantes evoluções na forma de identificação e tratamento das dependências entre requisitos de *software*. Essas classificações são importantes para o processo de priorização já que, por meio de sua avaliação, a equipe de desenvolvimento deve planejar os elementos constantes no próximo *release*. Essa tarefa facilita, por exemplo, a exclusão de requisitos conflitantes e/ou a inclusão de dependências hierárquicas.

Considerando os dados disponíveis nos documentos de especificação de requisitos cujo escopo desse trabalho contempla, o método proposto visa identificar dependências do tipo funcional entre os requisitos focando em relações do tipo *Constraint/Requiring* (ZHANG, 2013), e *Structural/Similar_to* (DAHLSTEDT; PERSSON, 2003). Nesses casos entende-se que é possível identificar esses *links* sem o auxílio de usuários, analisando as referências a classes candidatas descritas nos requisitos de um sistema.

2.4 PRIORIZAÇÃO DE REQUISITOS DE *SOFTWARE* (PRS)

Quando as expectativas do cliente são altas, os prazos são curtos e os recursos são limitados é necessário garantir que o produto contenha as funções mais essenciais (WIEGERS, 1999). O principal objetivo da PRS é auxiliar na escolha dessas funções em um sistema em desenvolvimento definindo estratégias e parâmetros capazes de fornecer um resultado mais apropriado para cada tipo de problema.

Para Bourque e Fairley (2014) a priorização de requisitos é necessária, não apenas como um meio de filtrar requisitos importantes, mas também para resolver conflitos e planejar entregas em etapas. Fato bastante comum nos dias de hoje com a adoção de métodos ágeis de desenvolvimento que buscam entregar *software* funcionando com frequência dando preferência a intervalos mais curtos. Para isso, as técnicas de PRS visam balancear o benefício do desenvolvimento de determinado requisito para o sistema com seu respectivo custo de implementação (WIEGERS, 1999).

Um dos problemas relacionados a essa tarefa está na dificuldade de obter informações reais que possam embasar tais decisões. Além disso, os requisitos geralmente dependem uns dos outros e as prioridades são relativas (BOURQUE; FAIRLEY, 2014).

Trata-se portanto de uma atividade subjetiva realizada pelo conjunto dos *stakeholders* do projeto, onde o papel do cliente é fornecer as informações sobre os requisitos mais importantes para ele, o que muitas vezes não é trivial para a equipe. Por outro lado, a equipe deve julgar os custos de implementação e a dificuldade técnica de cada requisito, pois possui melhores condições para isso (WIEGERS, 1999), podendo assim, planejar o próximo *release*.

Wiegers (1999) sugere manter a priorização o mais simples possível para auxiliar na tomada de decisões de desenvolvimento, selecionando um nível apropriado de abstração para a priorização dependendo da complexidade do problema, seja em nível de caso de uso, de recurso ou no nível detalhado do requisito funcional. O quadro a seguir, apresenta duas possíveis escalas de priorização desses artefatos de acordo com Wiegers (1999):

QUADRO 4: ESCALAS DE PRIORIZAÇÃO (WIEGERS, 1999).

Nome	Significado
Alto	Requisito crítico, necessário para o próximo *release*.
Médio	Requisito que suporta operações necessárias, mas pode esperar.
Baixo	Aprimoramento funcional ou de qualidade. Desejável dependendo dos recursos.
Essencial	O produto não é aceitável se esse tipo de requisito não for satisfeito.
Condicional	Melhoraria o produto, mas a falta não o torna inaceitável.
Opcional	Funções que podem ou não valer a pena.

FONTE: Adaptado de Wiegers (1999).

No entanto, mesmo que as prioridades devam ser julgadas com base em sua percepção de valor ao cliente ou importância, outros parâmetros implícitos devem ser levados em conta nesse processo. Isso pode ser percebido em Achimugu *et al*. (2014), que concluiu que as técnicas de priorização existentes sofrem de uma série de limitações que incluem, por exemplo, questões de dependência entre requisitos, falta de escalabilidade e coordenação entre as partes interessadas.

Nota-se assim, que requisitos podem ser classificados com grau maior ou menor de prioridade ao considerar suas interdependências. Por exemplo, um requisito do qual muitos outros do conjunto sejam dependentes, ou seja, com muitas relações do tipo *Requiring* (ZHANG, 2013), tende a ser classificado com grau superior a outros com poucas relações do tipo, independentemente da escala escolhida: alto, médio, baixo ou essencial, condicional, opcional (WIEGERS, 1999).

Diante desses e outros problemas, muitas técnicas de PRS têm sido propostas nos últimos anos. Em levantamento sobre o estado da arte em PRS, Achimugu *et al*. (2014) mapearam 49 técnicas distintas de priorização em 73 estudos selecionados, das quais mais da metade (26) foram utilizadas em no máximo 3 estudos, demonstrando que muitas tentativas de aperfeiçoamento para essa tarefa vêm sendo testadas.

Dentre as técnicas com mais citações na literatura, a AHP (*Analytic Hierarchy Process*) é a que mais aparece com 48 citações em Achimugu *et al*. (2014), porém, essa técnica implementa estratégia semelhante ao julgamento por comparações em pares para avaliar prioridades entre os requisitos, gerando grande quantidade de confrontações, o que resulta em problemas de escalabilidade (SHAO

et al., 2017) e, consequentemente, grande necessidade de esforço humano para analisar e julgar.

Recentemente, Shao *et al.* (2017) obtiveram sucesso nos experimentos realizados com seu método de priorização baseado em dependências e preferências dos *stakeholders*, os resultados demonstraram que levar em conta essas duas características pode resultar em melhorias na sequência final de prioridades quando comparada aos resultados de outras técnicas, inclusive a AHP. Mesmo assim, a técnica depende altamente de trabalho humano para ser realizada, já que as informações sobre dependências são informadas antecipadamente por desenvolvedores a partir de seus julgamentos e conhecimento empírico.

Diante das dificuldades citadas anteriormente como: a necessidade de identificar requisitos importantes, a avaliação do custo de implementação e dificuldade técnica dos requisitos, o impacto das relações de dependência e a necessidade de coordenação entre as partes interessadas no processo de priorização, este trabalho propõe uma nova técnica baseada em dependências entre requisitos identificadas com o auxílio de ferramentas de PLN. A correta identificação dessas dependências pode auxiliar na comunicação entre os interessados, além de amenizar o impacto das demais dificuldades citadas.

2.5 PROCESSAMENTO DE LINGUAGEM NATURAL (PLN)

Reese (2015, p. 2), define formalmente o PLN como um campo de estudo utilizando conceitos de informática, inteligência artificial e linguística formal para analisar a linguagem natural. Essas ferramentas podem contribuir para a solução de inúmeros problemas atuais pois, com a expansão da *internet*, temos visto grande crescimento na quantidade de dados não estruturados disponíveis. Ainda de acordo com Reese (2015, p. 3) elas podem ser aplicadas em textos que variam de algumas palavras de entrada do usuário para uma consulta da *internet* a vários documentos que precisam ser resumidos.

Para possibilitar a análise linguística de um texto em formato digital, é necessário definir claramente caracteres, palavras e sentenças em um documento, essa tarefa não é trivial, especialmente quando se considera a variedade de linguagens humanas e sistemas de escrita. Além disso, os sistemas de escrita frequentemente amplificam as ambiguidades constantes em uma linguagem,

tornando a resolução destas uma grande parte do desafio do PLN (INDURKHYA; DAMERAU, 2010, p. 9).

No entanto, com a evolução dos estudos na área, tornou-se possível analisar textos livres em diferentes níveis. Por representações sintáticas, analisando sua composição e estrutura ou semanticamente, por meio da identificação de relações entre conceitos, fato que, segundo Cambria e White (2014) tem recebido grande parte da atenção nos estudos atuais.

Nesse cenário, foram desenvolvidos alguns *frameworks* que suportam PLN em diferentes fases, entre eles estão o Apache OpenNLP, Stanford NLP, LingPipe, GATE (*General Architecture for Text Engineering*) e UIMA (*Unstructured Information Management Architecture*) (REESE, 2015). Este trabalho adotou o *framework* Apache OpenNLP, versão 1.9.1[4] por fornecer suporte às fases de PLN necessárias para cumprimento dos objetivos definidos e por contar com um bom suporte bibliográfico.

Essa seção tem por objetivo contribuir para uma melhor compreensão do tema apresentando os conceitos e ferramentas de PLN utilizados na análise de histórias de usuário submetidas como entradas do método proposto, enfatizando as fases de preprocessamento de texto, *part-of-speech tagging* e lematização.

2.5.1 Preprocessamento de texto

A fase inicial da PLN é chamada de preprocessamento de texto. De acordo com Indurkhya e Damerau (2010, p. 9) os caracteres, palavras e frases identificadas nesta etapa tornam-se as unidades fundamentais que serão passadas para as demais fases do processo. Esse passo contempla a triagem de documento, segmentação da mensagem e identificação de segmentos **interessantes** de texto (GALLIERS; JONES, 1993, p. 70, grifo nosso).

Inicialmente, para que um documento em linguagem natural seja interpretado por máquina, tornando-se legível é necessária a identificação do tipo de codificação de caracteres (INDURKHYA; DAMERAU, 2010, p. 9). Essa classificação, que ocorre na fase de triagem, serve principalmente para identificar os símbolos contidos no texto e adaptar a sequência de *bits* de caracteres do documento para um idioma conhecido por meio da aplicação de algoritmos.

---

4 *The Apache Software Foundation* (2017).

A classificação é necessária devido ao suporte das ferramentas a alfabetos diversos, possibilitando a leitura de vários idiomas. Bird *et al.* (2009, p. 94) complementam que algumas codificações (como ASCII e *Latin*-2) usam um único *byte* para codificação, suportando apenas um pequeno subconjunto de *Unicode*, o suficiente para um único idioma, enquanto outras (como UTF-8) usam vários *bytes*. Sendo assim, mediante a correta identificação da codificação do texto é possível interpretar a sequência de *bits* do documento, identificando a cadeia de caracteres e manipulando-os um a um.

Ainda durante a triagem de um documento, são realizados procedimentos para extrair como saída um bloco de texto puro e bem definido. Esses procedimentos ignoram elementos auxiliares como tabelas, formatações, e *tags* (INDURKHYA; DAMERAU, 2010, p. 10), se fazem necessários pois, nem sempre os documentos preprocessados possuem apenas texto em seu conteúdo, mas pode ser desejável que apenas isso permaneça para observações posteriores. Por exemplo, em um arquivo de texto elaborado em formato XML, as *tags* (campos) podem ser ignoradas durante o preprocessamento, extraindo apenas seus conteúdos.

Após a extração do texto puro e bem definido, são identificadas as palavras e sentenças constantes na etapa chamada de segmentação de texto, dividindo a sequência de caracteres em busca de separadores como espaços e elementos de pontuação, respeitando as regras do idioma de entrada. Para este trabalho, o idioma de entrada selecionado foi o inglês.

De acordo com Reese (2015), para a maioria dos textos em inglês, o espaço em branco é usado como um delimitador. Esse tipo de delimitador geralmente inclui além de espaços em branco, tabulações e caracteres de nova linha. Esse procedimento, conhecido como tokenização, interpreta a cadeia de caracteres obtida após a triagem gerando como saída uma sequência de palavras, denominadas *tokens*.

Como exemplo em Reese (2015, p. 14), considerando a *string* "*Mr. Smith went to 123 Washington avenue*." e um processo de tokenização simples cujo caractere separador seja o espaço, o resultado da extração de *tokens* será o seguinte:

Mr.

Smith

went
to
123
Washington
avenue.

A saída da tokenização pode ser útil para verificadores ortográficos, processamentos de pesquisa simples e também para outras tarefas do PLN como a identificação de partes do discurso (*part-of-speech tagging*), sendo frequentemente apenas um passo em uma sequência maior de tarefas (REESE, 2015, p. 32).

Finalmente, a sequência de *tokens* é analisada de modo reagrupar palavras em frases. Bird *et al*. (2009, p. 112) consideram que manipular textos ao nível de palavras individuais pressupõe muitas vezes a capacidade de dividir um texto em frases individuais. Essas frases, também chamadas de sentenças, são unidades de texto mais longas que as palavras normalmente delimitadas por sinais de pontuação.

Obter sentenças a partir dos *tokens* possibilita um novo nível de interpretação para futuros procedimentos que busquem extrair dados mais relevantes sobre eles, considerando o contexto e a estrutura das sentenças onde estão inseridos. Logo, a análise semântica mencionada anteriormente não pode ser realizada sem que essas frases sejam identificadas, pois pressupõe que o contexto fornece dados que devem ser considerados. Esse passo também é determinante para a próxima fase, a marcação de partes do discurso (*part-of-speech tagging*).

Sendo assim, obtém-se a saída desejada depois da conclusão de todas as etapas do preprocessamento, qual seja, o conjunto de sentenças na forma de frases puras, extraídas do documento inicial. No contexto deste trabalho, o preprocessamento de texto visa facilitar as próximas fases do PLN, fornecendo esses elementos como entrada para a fase seguinte.

2.5.2 *Part-of-speech (POS) Tagging*

O termo POS *tagging* pode ser traduzido para o português como marcação de parte do discurso. Marcação (*tagging*) é o processo de atribuir uma descrição, chamada de *tag*, a cada um dos *tokens* das sentenças identificadas no preprocessamento de texto.

Normalmente referem-se ao tipo de palavra representada pelo *token* (REESE, 2015, p. 125), adicionando uma espécie de etiqueta de classificação a eles. O conjunto básico de categorização léxica consiste em substantivos, verbos e adjetivos, porém frequentemente são adotadas categorias e/ou subcategorias adicionais por outros modelos linguísticos.

Para Bird *et al*. (2009, p. 179), as categorias de palavras são úteis para muitas tarefas relacionadas ao PLN. Geralmente, surgem da simples análise de distribuição de palavras no texto, já que a marcação de adjetivos, por exemplo, frequentemente escritos da mesma forma que substantivos, é feita observando entre outras regras, a de posicionamento (SCHWEINBERGER, 2016). Portanto, cabe à ferramenta escolhida aplicar algoritmos para, a partir dos *tokens* identificados, adicionar as etiquetas com maiores probabilidades de classificação.

Em algumas sentenças a tarefa parece trivial, no entanto, nos casos em que as palavras são ambíguas, possuindo papéis e significados diferentes com a mesma morfologia, isso possibilita a associação de duas ou mais *tags* diferentes a elas e, escolher qual melhor se aplica, depende da estratégia implementada em cada *framework*.

Ante essa dificuldade, Reese (2015) afirma que um dicionário nem sempre é capaz de determinar a *tag* POS de um *token*. A título de exemplo, o significado de palavras como *bill* e *force* dependem do contexto. A frase a seguir demonstra como eles podem ser usados na mesma sentença com papéis diferentes.

"*Bill used the force to force the manager to tear the bill in two.*"

Pode-se observar que a primeira ocorrência da palavra *bill* refere-se a um nome próprio (Bill), enquanto a segunda a um substantivo (conta). Já a primeira ocorrência da palavra *force* refere-se a um substantivo (força), enquanto a segunda a um verbo (forçar) (REESE, 2015, p. 128). Constata-se que nos dois casos as palavras devem ser marcadas com uma *tag* diferente em cada ocorrência. Para possibilitar tal marcação, os *frameworks* de PLN possuem uma lista de *tags* possíveis para atribuição. No Quadro 5 é apresentada a lista de *tags* do Apache OpenNLP conforme Schweinberger (2016) respeitando a classificação dos termos no idioma inglês.

QUADRO 5: LISTA DE *TAGS* DO *FRAMEWORK* OPENNLP

Part–of–Speech Tag	***Part–of–Speech category***
CC	*Coordinating conjunction*
CD	*Cardinal number*
DT	*Determiner*
EX	*Existential there*
FW	*Foreign word*
IN	*Preposition or subordinating conjunction*
JJ	*Adjective*
JJR	*Adjective, comparative*
JJS	*Adjective, superlative*
LS	*List item marker*
MD	*Modal*
NN	*Noun, singular or mass*
NNS	*Noun, plural*
NNP	*Proper noun, singular*
NNPS	*Proper noun, plural*
PDT	*Predeterminer*
POS	*Possessive ending*
PRP	*Personal pronoun*
PRP$	*Possessive pronoun*
RB	*Adverb*
RBR	*Adverb, comparative*
RBS	*Adverb, superlative*
RP	*Particle*
SYM	*Symbol*
TO	*to*
UH	*Interjection*
VB	*Verb, base form*
VBD	*Verb, past tense*
VBG	*Verb, gerund or present participle*
VBN	*Verb, past participle*
VBP	*Verb, non–3rd person singular present*
VBZ	*Verb, 3rd person singular present*
WDT	*Wh–determiner*
WP	*Wh–pronoun*
WP$	*Possessive wh–pronoun*
WRB	*Wh–adverb*

FONTE: SCHWEINBERGER (2016).

Conforme demonstrado em Reese (2015), considerando a sentença "*The voyage of the Abraham Lincoln was for a long time marked by no special incident.*" e

a tabela de *tags* do *framework* OpenNLP exibida anteriormente, o processo de marcação para a sentença retorna o seguinte resultado:

The/DT *voyage*/NN *of*/IN *the*/DT Abraham/NNP Lincoln/NNP *was*/VBD *for*/IN *a*/DT *long*/JJ *time*/NN *marked*/VBN *by*/IN *no*/DT *special*/JJ *incident.*/NN.

O *framework* permite vincular à atribuição de cada *tag* uma probabilidade que indica a chance de a atribuição estar correta. Para a situação supracitada, foram geradas as seguintes probabilidades:

DT/0.992 NN/0.990 IN/0.989 DT/0.990 NNP/0.996 NNP/0.991 VBD/0.994 IN/0.996 DT/0.996 JJ/0.991 NN/0.994 VBN/0.860 IN/0.985 DT/0.960 JJ/0.919 NN/0.832. (REESE, 2015, p. 134).

Observa-se que as menores probabilidades (0.86 e 0.832) foram associadas às palavras *marked* e *incident*, respectivamente. Ambas podem assumir mais de uma *tag* dependendo do contexto. *Marked,* assinalada como VBN (verbo no particípio passado) em alguns casos poderia assumir a *tag* JJ, pois também pode ser um adjetivo. O mesmo ocorre com *incident*, assinalado com a *tag* NN (substantivo) que também poderia assumir a *tag* JJ, porém a estratégia implementada por essa funcionalidade do *framework*, atribui ao *token* em questão a *tag* de maior valor de probabilidade, configurando a marcação exibida.

A POS *tagging* é de fundamental importância para este trabalho, pois possibilita a identificação de termos que possam representar classes candidatas no contexto dos problemas submetidos, percorrendo a lista de *tokens* extraídos, em busca dos assinalados com *tags* que se refiram a substantivos (NN, NNS, NNP, NNPS). Essas classes candidatas são utilizadas na averiguação de dependências nas próximas etapas do método.

2.5.3 Lematização (*lemmatization*)

Bird *et al.* (2009, p. 121) definem a lematização como um processo que mapeia as várias formas de uma palavra para a forma canônica ou de citação da palavra, também conhecida como lexema ou lema. Para Indurkhya e Damerau (2010, p. 31), trata-se de uma tarefa básica da análise léxica relacionar as variantes morfológicas ao seu lema, que se encontra em um dicionário, agrupado com sua informação semântica e sintática invariante.

Isto é, o processo de lematização busca agrupar termos com escritas diferentes, que se referem à mesma palavra, ou seja, que possuem a mesma raiz em um termo único mais genérico, ignorando assim, variações de plural, tempos verbais, *etc*.

Evocando o exemplo da seção anterior, que simulou a marcação de partes do discurso com a frase "*The voyage of the Abraham Lincoln was for a long time marked by no special incident.*", temos que a palavra *was*, assinalada como VBD (verbo no passado) corresponde ao lema *be*, sua forma mais genérica, no infinitivo. O mesmo acontece com *marked* que, por ter sido classificado como VBN, assumiria sua forma mais genérica, *mark*. Se a utilização de substantivos for feita no plural (*e.g.*, *women*), o lema relacionado é o termo no singular (*woman*).

Para Indurkhya e Damerau (2010, p. 31), dado que um lema tem semântica invariante, encontrar uma ocorrência de uma das suas variantes morfológicas satisfaz as exigências semânticas de uma busca, além disso, a morfologia é usada para criar palavras através de derivação.

Diante do exposto, utilizar a lematização como parte do processamento de textos em linguagem natural possibilita lidar com um conjunto menor, porém mais significativo de termos, evitando algumas ambiguidades. Para este trabalho a lematização permite flexibilizar a escrita das histórias de usuário de acordo com as regras da linguagem às quais estamos acostumados, sem a necessidade de adotar padrões muito rígidos de especificação além do formato adotado. A generalização de termos em lemas mais significativos evita a geração de classes candidatas ambíguas, melhorando a qualidade dos resultados obtidos.

2.6 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou os seguintes conceitos: requisitos de *software*, desenvolvimento ágil, histórias de usuário, interdependência entre requisitos, PRS, PLN, preprocessamento de texto, POS *tagging* e lematização. A integração desses conceitos possibilita o entendimento do método de priorização denominado Dependency Rank. Seu modo de funcionamento, arquitetura e principais rotinas implementadas são aprofundados no Capítulo 4.

3 TRABALHOS CORRELATOS

O presente capítulo apresenta os principais trabalhos relacionados ao tema proposto. Foram avaliadas publicações na área de PRS que utilizaram as relações de dependência entre requisitos em sua definição de prioridades, explorando as principais fontes de pesquisa, a evolução de estudos da área na última década e as principais limitações encontradas para o contexto deste trabalho.

3.1 METODOLOGIA DO MAPEAMENTO

De acordo com Kitchenham e Charters (2007 citado por PETERSEN *et al.*, 2015, tradução nossa), "estudos de mapeamento sistemático ou estudos de escopo são projetados para dar uma visão geral de uma área de pesquisa através da classificação e contagem de contribuições em relação às categorias dessa classificação". Para Petersen *et al.* (2015), esse tipo de estudo possui a característica de consolidar o conhecimento sobre os bons princípios nas publicações atuais.

Esse capítulo adota a metodologia de mapeamento sistemático conforme Petersen *et al.* (2015), buscando apresentar um panorama das pesquisas atualmente publicadas na área de PRS que investiguem os impactos das relações de dependência entre requisitos em sua abordagem.

Os resultados desse mapeamento auxiliam no entendimento da arquitetura do método proposto, evidenciando as principais fontes de consulta e diferenças entre escopos, técnicas, tratamento de dependências e limitações identificadas nesse processo.

Esse mapeamento explorou as bibliotecas digitais IEEE Xplore, ACM *Digital Library*, ScienceDirect, Springer e Google *Scholar.* Em razão da pouca quantidade de trabalhos publicados em língua portuguesa nessa área, as *strings* de busca foram definidas apenas em inglês.

3.1.1 Questões de pesquisa

Com foco em atingir os objetivos do mapeamento sistemático proposto no trabalho, as questões de pesquisa elaboradas intentam identificar estudos sobre as técnicas de PRS publicadas na última década (a partir do ano de 2008),

investigando como identificam as dependências entre requisitos e quais as principais limitações e canais de publicação encontrados. Essa delimitação se deve ao crescente número de publicações na área desde então, conforme observado em Achimugu *et al*. (2014). Sendo assim, foram elaboradas as seguintes questões:

a) Q1: Quais técnicas (métodos) de PRS têm sido propostas levando em conta as interdependências entre requisitos?
b) Q2: Como as interdependências são identificadas por essas técnicas (métodos)?
c) Q3: Quais são as limitações das técnicas (métodos) encontradas?
d) Q4: Qual a fonte e o ano de publicação desses artigos?

3.1.2 Estratégia de Busca

Como estratégia de busca foi adotado parcialmente o critério PICOC (*Population*, *Intervention*, *Comparison*, *Outcome*, *Context*) conforme detalhado por Brereton *et al*. (2007), focando em população (*population*) e intervenção (*intervention*) e resultado (*outcome*).

A respeito da população, foram consideradas as publicações na área de PRS encontradas nas bibliotecas digitais citadas anteriormente, sendo extraído o termo "*software requirements prioritization*" como *string* de busca inicial. Como intervenção foram consideradas as técnicas ou métodos de PRS propostos nos estudos, que respondem à questão Q1. Finalmente, o critério resultado gerou o termo "*requirements interdependencies*", no intuito de filtrar resultados com esse enfoque no cálculo de prioridades.

Como não se buscou comparação entre intervenções, ou contextos de aplicação específicos, os critérios comparação (*comparison*) e contexto (*context*) não foram utilizados. A análise inicial dos resultados levou em consideração o texto contido nos campos título, subtítulo e resumo (*abstract*) de cada estudo, sendo assim, cada ocorrência que atendeu às regras definidas passou para a fase de seleção dos resultados.

A técnica *snowballing* (WOHLIN, 2014) foi aplicada para identificar outros estudos de interesse ao consultar a lista de referências dos artigos retornados pelas *strings* de busca desse mapeamento.

A partir dos termos de busca inicias as *strings* foram adaptadas para tornar os resultados mais abrangentes, adicionando termos alternativos equivalentes combinados com os operadores booleanos de conjunção e disjunção, quando aceitos pelos respectivos mecanismos.

Para o termo "*software requirements prioritization*", considerou-se "*software*" como termo obrigatório e os termos alternativos "*requirement prioritization*", "*requirement priority*" e "*prioritizing requirements*". Para "*requirements interdependencies*", uma vez que a palavra "*requirement*" e seus equivalentes já foi abrangida pela *string* anterior, considerou-se apenas equivalências de "*interdependencies*", sendo assim, foram adicionados "*dependency*", "*dependencies*" e "*dependent*". O Quadro 6 exibe as *strings* de busca executadas por biblioteca digital. A pesquisa foi atualizada no mês de outubro de 2018.

QUADRO 6: *STRINGS* PESQUISADAS POR BIBLIOTECA DIGITAL.

Biblioteca	***String* de busca**
ACM *Digital Lybrary*	*content.ftsec:(+software +"requirements prioritization" "requirement prioritization" "requirements priority" "prioritizing requirements") AND recordAbstract:("dependency" "dependencies" "dependent" "interdependencies")*
Google *Scholar*	*software AND ("requirements prioritization" OR "requirement prioritization" OR "requirements priority" OR "prioritizing requirements") AND ("dependency" OR "dependencies" OR "interdependencies") -Springer -Sciencedirect -ieee -dl.acm*
IEEE *Explorer*	*"software" AND (("requirement prioritization" OR "requirements prioritization" OR "prioritizing requirements" OR "requirement priority") AND ("dependency" OR "dependencies" OR "dependent" OR "interdependencies"))*
ScienceDirect	*"requirements prioritization" or "requirement prioritization" or "requirements priority" or "prioritizing requirements"*
Springer	*'software AND requirements AND prioritization AND (dependency OR dependencies OR dependent OR interdependencies)' within Computer Science*

FONTE: O autor (2018).

3.1.3 Critérios de seleção

O processo de seleção foi realizado em duas etapas. Inicialmente foram avaliados títulos, subtítulos e resumos de 333 resultados retornados pelas buscas em bibliotecas digitais. Nos casos excepcionais, em que após avaliar esses itens a inclusão ou exclusão do resultado não ficou clara, foi realizada a leitura completa de alguns textos. Com a finalidade de filtrar os mais relevantes, foram aplicados filtros

definidos por critérios de inclusão e exclusão, adaptados ao contexto desse mapeamento, conforme exemplo em Achimugu *et al*. (2014).

Critérios de inclusão:

a) Artigos em que o objeto principal de estudo foi a PRS;
b) Artigos que abordaram uma ou mais técnicas (métodos) de PRS;
c) Artigos que utilizaram a dependência entre requisitos como um parâmetro de priorização;
d) Artigos publicados a partir do ano de 2008 até o ano de 2018.

Critérios de exclusão:

a) Artigos cujo objeto principal não foi a PRS ou não utiliza a dependência entre requisitos como parâmetro de avaliação;
b) Estudos de revisão sistemática de literatura, pois não é possível obter muitos detalhes sobre as técnicas constantes neles;
c) Estudos apresentados em idiomas diferentes do idioma inglês;
d) Estudos cujo texto completo não foi acessível por restrições autorais;
e) Estudos duplicados.

Após aplicação dos referidos filtros foram obtidos 17 artigos, dos quais 12 eram acessíveis, o resultado das buscas é exibido na Tabela 1.

TABELA 1: RESULTADOS DAS BUSCAS E APLICAÇÃO DOS FILTROS

Biblioteca	Resultados	Filtrados	Acessíveis
ACM *Digital Lybrary*	11	4	4
Google *Scholar*	277	2	0
IEEE Xplore	12	6	6
ScienceDirect	18	3	2
Springer	15	2	0
Totais	**333**	**17**	**12**

FONTE: O autor (2018).

3.1.4 Extração de dados

Os estudos selecionados passaram para a etapa de extração de dados, tal etapa visa sintetizar as informações relevantes dos artigos filtrados fundamentadas nos critérios escolhidos para esse trabalho, os itens extraídos foram gerados a partir

dos objetivos e questões de pesquisa detalhados anteriormente, conforme modelo no Quadro 7.

QUADRO 7: FORMULÁRIO DE EXTRAÇÃO DE DADOS

Item	Valor	Questão
ID	Valor inteiro	-
Título	Nome do artigo	-
Autor	Nomes dos autores	-
Ano	Ano de publicação	Q4
Fonte	ACM-DL, Google *Scholar*, IEEE *Xplore*, ScienceDirect, Springer.	Q4
Técnica de PRS	AHP (Processo Analítico Hierárquico), Baseada em dependências, Baseada em restrições, Baseada em valores, Otimização por Algoritmos, *Ranking*, Teste de Mutação.	Q1
Tipo de detecção de dependências	Guiada, Automática.	Q2
Limitações	Alta demanda de trabalho humano, Escalabilidade, Poucos critérios de priorização, Requisitos incompletos.	Q3

FONTE: O autor (2018).

Os valores referentes às técnicas de priorização e limitações foram adaptados para os itens citados nas publicações acessíveis. Dentre os tipos de detecção de dependências, esta pesquisa visou classificar entre guiada (quando é informada por trabalho humano) ou automática (reconhecida por procedimento computacional).

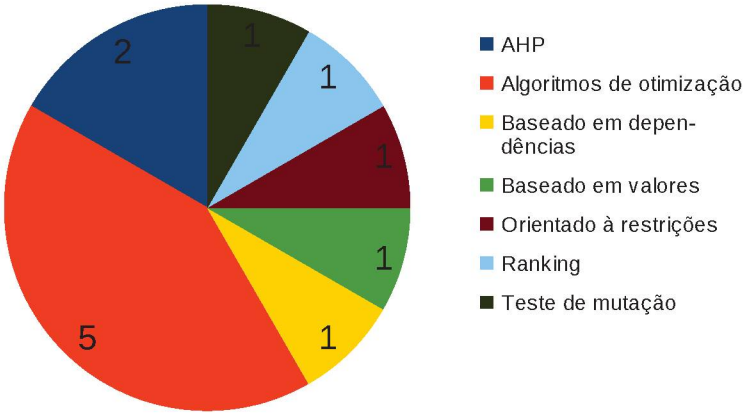
3.2 RESULTADOS DO MAPEAMENTO

Os resultados do mapeamento encontram-se detalhados e compilados e estão disponíveis para consulta na *internet*[5].

Após análise dos 12 artigos selecionados, os resultados foram organizados com o propósito de responder as questões de pesquisa apresentadas na Seção 3.1.1. O Gráfico 1 apresenta o total de publicações encontradas, agrupadas por técnica de priorização.

5 Resultados disponíveis no endereço: https://bit.ly/2Hr0ixs

GRÁFICO 1: RESULTADOS DA QUESTÃO Q1.



FONTE: O autor (2018).

Em resposta à questão Q1, foram encontradas 7 técnicas diferentes nos 12 artigos selecionados. A técnica mais citada foi a priorização por algoritmos de otimização, que constou em 5 das 12 publicações. O processo analítico hierárquico (AHP) foi abordado em 2 publicações. Outras técnicas encontradas foram: baseada em dependências, baseada em valores, orientada à restrições, *ranking* e teste de mutação.

A técnica AHP utiliza comparação entre pares de requisitos, recebendo avaliações de usuários humanos para embasar sua definição de prioridades (OGNJANOVIC *et al.*, 2011), (ALAWNEH, 2018). Assim como a AHP, as técnicas baseadas em valores (FITSILIS *et al.*, 2010) ou restrições (REGNELL; KUCHCINSKI, 2011) recebem de usuários humanos os valores que populam suas variáveis para, posteriormente, definir seus grupos de prioridades. Portanto, a principal diferença entre essas técnicas diz respeito aos critérios de avaliação.

Os algoritmos de otimização são aplicados a métodos de PRS em contextos em que algumas variáveis são ponderadas por algoritmos buscando a melhor sequência de priorização. Cada estratégia implementa seu conjunto de variáveis. Como exemplo, Li *et al*. (2017) utilizam as variáveis importância, dependência, custo de implementação e probabilidade de superação de custo em seu algoritmo. Enquanto Tonella *et al*. (2013) utilizam apenas as dependências combinadas com preferências informadas por combinação em pares em sua abordagem.

A técnica baseada em dependências desenvolvida por Alzyoudi *et al*. (2015), recebe de usuários as relações de dependências entre requisitos, gerando

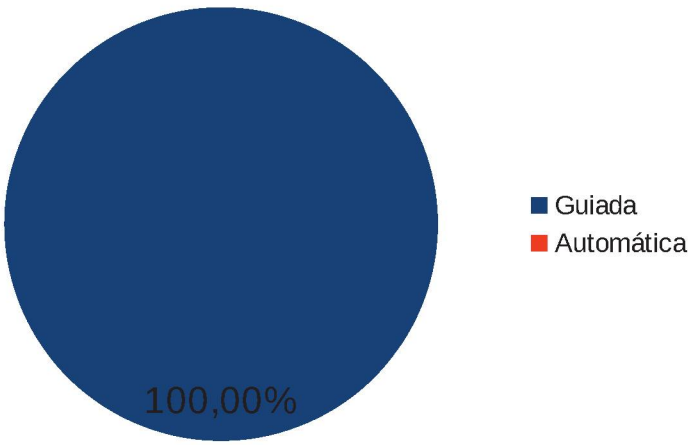
incrementos de *software* para um novo lançamento, ou seja, consiste em uma técnica similar à baseada em restrições, apresentada por Regnell e Kuchcinski (2011). As relações de dependência também compõem um parâmetro na abordagem de Shao *et al*. (2017), que utilizam além disso, as preferências dos usuários, identificadas por comparações em pares de forma análoga à AHP, para definição de seu *ranking*.

Por fim, Condori-Fernandez *et al*. (2018) propuseram a aplicação de conceitos de teste de mutação, amplamente usados na área de teste de *software* para validar sequências de priorização relacionadas a requisitos não-funcionais.

Com base no exposto, é possível perceber que a quantidade de variáveis que implicam na definição das prioridades em um processo de PRS faz com que várias possibilidades sejam testadas visando atingir uma sequência correta. Ainda que os dados de entrada das técnicas citadas possam ser entendidos como precisos, por serem informados por usuários, pode-se notar a excessiva necessidade de trabalho humano envolvida.

A questão Q2 investigou o tipo de detecção de dependências entre requisitos abordado nos estudos. Sua principal finalidade foi identificar se alguma publicação selecionada tentou realizar essa detecção de forma automática e, em caso de resposta positiva, como a fez. No entanto, nenhuma publicação entre as selecionadas automatizava a detecção de dependências. O resultado é apresentado no Gráfico 2.

GRÁFICO 2: RESULTADO DA QUESTÃO Q2.

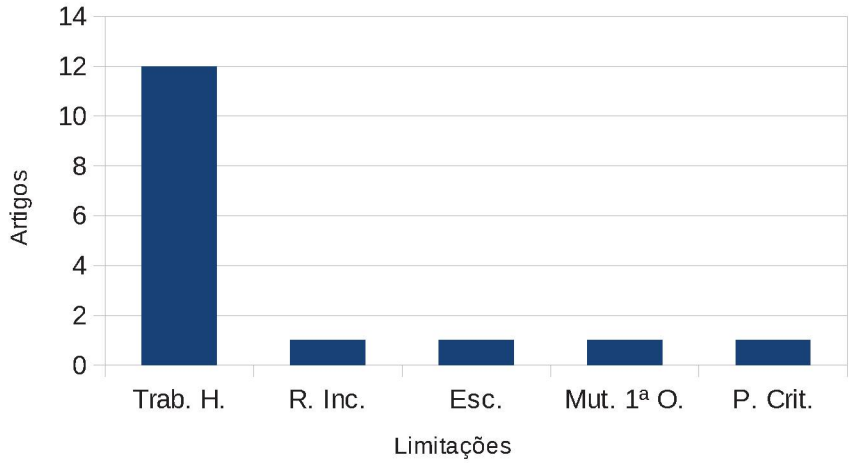


FONTE: O autor (2018).

Acerca das principais limitações dos estudos, foco da questão Q3, são citadas limitações de escalabilidade, a utilização de poucos critérios de priorização,

dificuldade em priorizar requisitos incompletos e, principalmente, a alta demanda por trabalho humano. Os resultados da questão são apresentados no Gráfico 3.

GRÁFICO 3: RESULTADO DA QUESTÃO Q3



FONTE: O autor (2018).

A alta demanda por trabalho humano (Trab. H.) foi citada ou ficou implícita em todos os 12 artigos. Essa limitação refere-se à necessidade de um usuário popular as variáveis empregadas pelas técnicas na análise de suas prioridades.

As demais limitações identificadas foram citadas em apenas 1 trabalho cada, sendo:

- R. Inc.: Abordagem não contempla requisitos incompletos (OGNJANOVIC *et al.*, 2011);
- Esc.: Problemas de escalabilidade (REGNELL; KUCHCINSKI, 2011);
- Mut. 1ª O.: Utilização de mutantes apenas de primeira ordem (CONDORI-FERNANDEZ *et al.*, 2018);
- Utilização de poucos critérios de priorização (ALZYOUDI *et al.*, 2015).

Pode-se perceber que a necessidade de identificar os critérios de priorização manualmente, seja por atribuição de valores, restrições, comparação em pares, *etc.*, faz com que as técnicas abordadas nos estudos possuam como desvantagem a alta demanda por trabalho humano, bem como acontece com as técnicas de priorização. Essa necessidade é potencializada pela forma de identificação de dependências, feita em todos os casos de maneira guiada.

Os resultados sobre a questão Q4 são apresentados no Quadro 8. Nele, o resultado do mapeamento foi simplificado, focando nos resultados obtidos por fonte e ano.

QUADRO 8: PUBLICAÇÕES POR FONTE E ANO.

Fonte	Ano	Título	Autor
IEEE Xplore	2017	*Search-Based Uncertainty-Wise Requirements Prioritization*	Yan Li; Man Zhang; Tao Yue; Shaukat Ali; Li Zhang
IEEE Xplore	2016	*Planning Optimal Agile Releases via Requirements Optimization*	Joseph Gillain; Ivan Jureta; Stéphane Faulkner
IEEE Xplore	2014	*Requirements Prioritization and Next-Release Problem under Non-additive Value Conditions*	Ashish Sureka
IEEE Xplore	2011	*Exploring Software Product Management decision problems with constraint solving - opportunities for prioritization and release planning*	Björn Regnell; Krzysztof Kuchcinski
IEEE Xplore	2010	*Supporting the Requirements Prioritization Process Using Social Network Analysis Techniques*	Panos Fitsilis; Vassilis Gerogiannis; Leonidas Anthopoulos; Ilias K. Savvas
IEEE Xplore	2010	*Using Interactive GA for Requirements Prioritization*	Paolo Tonella; Angelo Susi; Francis Palma
ACM *Digital Library*	2018	*Towards a functional requirements prioritization with early mutation testing*	Nelly Condori-Fernandez; Maria Fernanda Granda; Tanja E. J. Vos
ACM *Digital Library*	2015	*A Probability Algorithm for Requirement Selection In Component-Based Software Development*	Ruba Alzyoudi; Khaled Almakadmeh; Hutaf Natoureah
ACM *Digital Library*	2014	*Applying search algorithms for optimizing stakeholders familiarity and balancing workload in requirements assignment*	Tao Yue; Shaukat Ali
ACM *Digital Library*	2011	*Conditional preferences in software stakeholders' judgments*	Ivana Ognjanovic; Dragan Gašević; Ebrahim Bagheri; Mohsen Asadi
ScienceDirect	2017	*DRank: A semi-automated requirements prioritization method based on preferences and dependencies*	Shao Fei; Peng Rong; Lai Han; Wang Bangchao
ScienceDirect	2013	*Interactive requirements prioritization using a genetic algorithm*	Paolo Tonella; Angelo Susi; Francis Palma

FONTE: O autor (2018).

3.3 ESTUDOS COM MAIOR PROXIMIDADE AO TEMA DA PESQUISA

Os estudos de Shao *et al*. (2017) e Alzyoudi *et al*. (2015) possuem maior proximidade a esta pesquisa por considerarem as dependências entre requisitos

como critério de primeira importância para priorização em suas respectivas abordagens.

Shao *et al*. (2017) apresentaram o método DRank baseado em preferências e dependências para a priorização de requisitos de *software*. O método desenvolvido visa utilizar essas dependências para melhorar a qualidade da priorização de acordo com as preferências dos *stakeholders* de um projeto, assim como diminuir a carga de trabalho humano e a necessidade de envolvimento de engenheiros de requisitos muito experientes.

Os resultados dos experimentos foram comparados a outras técnicas de PRS, obtendo melhoras em relação ao consumo de tempo. O estudo também concluiu que contemplar as dependências pode melhorar a precisão da sequência final de priorização. As principais limitações citadas no estudo dizem respeito à dificuldade de identificação de dependências de forma mais rápida e completa.

O método DRank se aproxima do método proposto neste trabalho em seus objetivos e critérios de priorização, possuindo como diferença o julgamento de preferências por comparação entre pares e o mapeamento guiado de dependências. Esse julgamento tende a impactar positivamente na sequência de priorização obtida pelo DRank, no entanto, espera-se que o método proposto, detalhado no Capítulo 4, alcance resultados aproximados com menor necessidade de tempo/esforço.

Alzyoudi *et al*. (2015) propuseram um método de priorização de requisitos funcionais para o modelo incremental de desenvolvimento de *software* baseado nos relacionamentos de dependência entre os requisitos de um sistema. A partir das indicações de dependências fornecidas por *stakeholders*, uma matriz é preenchida com as restrições de implementação, adaptadas das definições de dependência apresentadas em Zhang (2013).

Dentre as restrições, "*Support*(R_i, R_j)" indica que um requisito só pode ser implementado após seu suporte, "*Contradict*(R_i, R_j)" que dois requisitos não podem ser implementados no mesmo incremento e, por fim, "*Before*(R_i, R_j)" que sugere que a implementação de um requisito o mais rápido possível após o outro constante na relação pode fornecer vantagem para o referido processo de *software*. O algoritmo de priorização elaborado explora a matriz de dependências agrupando requisitos em forma de incrementos, simplificando o processo de seleção e priorização (ALZYOUDI *et al*., 2015).

Ainda que o método proposto por Alzyoudi *et al*. (2015) busque identificar incrementos de *software* por meio de dependências para depois submetê-los a um processo de priorização, a estratégia de relacionar requisitos dependentes se assemelha à proposta neste trabalho. Assim como em Shao *et al*. (2017), o mapeamento de dependências é feito de maneira guiada, possibilitando fornecer detalhes dessas relações informando o tipo de restrição. Portanto, espera-se que o método proposto alcance valores menos precisos, porém aproximados e com menor necessidade de esforço.

Por considerar que a extração de requisitos a partir de texto livre é parte fundamental para este trabalho, foram pesquisados, complementarmente, artigos cujo objetivo principal foi reconhecer requisitos de *software* utilizando ferramentas de PLN.

A pesquisa foi feita por busca pela *string* exibida a seguir na biblioteca digital Google *Scholar*, sendo atualizada no mês de outubro de 2018.

String de busca: *software AND (("feature" OR "requirement") AND "extraction") AND ("natural language processing").*

Os estudos considerados mais relevantes por possuírem maior proximidade com o tema da pesquisa em questão foram:

- Deshpande (2012) propôs uma ferramenta para automatizar a geração de diagramas de classes da UML a partir de descrições de problemas ou especificações de requisitos em linguagem natural. A ferramenta desenvolvida utilizou o *toolkit* OpenNLP *Parser* para as tarefas relacionadas ao processamento da linguagem e o dicionário WordNet (PRINCETON UNIVERSITY, 2018) para possibilitar relações de generalização entre substantivos, evitando ambiguidades.

  Os textos submetidos são percorridos em busca de classes candidatas, chamados de conceitos, identificados por substantivos. Posteriormente esses conceitos são relacionados, gerando um diagrama de classes do problema. De acordo com o autor, a ferramenta mostrou-se eficaz para o estudo de caso realizado.
- Em Sree-Kumar *et al*. (2018) foi proposto um *framework* chamado *FeatureX* para extração de características e relações, construída sobre ferramentas de

PLN com o objetivo de identificar recursos (*features*) e os relacionamentos entre eles.

O *framework* aborda as principais limitações de trabalhos anteriores na área. É dividido em três módulos sendo (1) um analisador léxico para o preprocessamento de texto e extração de entidades, (2) um módulo de aprendizado de máquina para identificar e classificar recursos candidatos e (3) um módulo de relações com as heurísticas para categorização dos relacionamentos. A abordagem apresentou melhorias com relação ao *recall* pela diminuição de falsos positivos em relação aos métodos propostos anteriormente, obtendo 90% de acerto entre os recursos e 45% entre os relacionamentos.

3.4 CONSIDERAÇÕES DO CAPÍTULO

Atualmente, existe grande variedade de técnicas disponíveis para auxiliar na tarefa de PRS, fato que pode ser observado, por exemplo, em Achimugu *et al.* (2014) e Pitangueira *et al.* (2015). A partir da análise dos trabalhos correlatos citados, é possível inferir que variáveis como o tipo do problema abordado, os recursos e a equipe disponível podem influenciar na escolha dessas técnicas.

Os relacionamentos de dependência também costumam ser definidos sem seguir um modelo padrão. Sendo assim, nota-se que apesar dos benefícios diante da utilização das informações de interdependências para definição de prioridades entre requisitos, ainda não há um consenso sobre boas práticas no mapeamento delas, sendo uma área com potencial para inovações.

Já a extração de recursos ou requisitos auxiliada por PLN tem sido feita de forma menos variada, partindo geralmente da identificação de conceitos relevantes para o contexto durante as fases de preprocessamento, utilizando analisadores léxicos e sintáticos e, por vezes, com auxílio de dicionários de sinônimos e camadas adicionais que objetivam evitar ambiguidades e complementar o entendimento sobre os termos encontrados.

Diante disso, este trabalho propõe um método de PRS com classificação por *ranking*, similar ao utilizado em Shao *et al.* (2017). Sua meta é gerar uma sugestão de *ranking* de requisitos intuitivo, que possa ser facilmente alterado, entendido e reproduzido manualmente por desenvolvedores, de modo a simplificar a participação

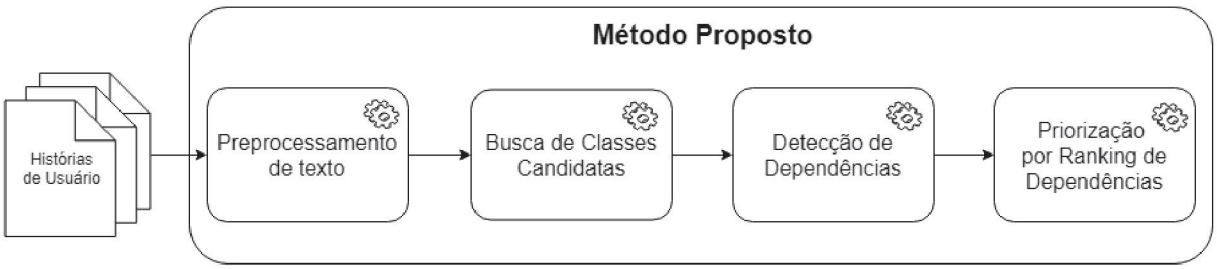
de profissionais no experimento e a comparação dos resultados. A principal diferença entre o ranqueamento proposto neste trabalho e o proposto por Shao *et al*. (2017) está na automatização do processo que pré-classifica os requisitos a partir de suas dependências, sem considerar preferências de *stakeholders* nessa fase.

As relações de interdependência investigadas pelo presente trabalho são mapeadas considerando os tipos *Constraint/Requiring,* apresentado em Zhang (2013) e *Structural/Similar_to*, de Dahlstedt e Persson (2003), tendo como principal diferencial o meio de obtenção automatizado. Para isso, textos que contém às descrições de requisitos de um determinado problema são processados por analisadores léxicos e sintáticos do *framework* Apache OpenNLP em busca de classes candidatas, similarmente a Deshpande (2012).

4 MÉTODO DEPENDENCY RANK

O presente capítulo descreve a arquitetura do método proposto, visa definir seus parâmetros de funcionamento e detalhar as camadas e atividades de processamento necessárias desde sua entrada até a obtenção da saída. Uma visão geral é exibida na Figura 1.

FIGURA 1: ARQUITETURA GERAL DO DEPENDENCY RANK



FONTE: O autor (2018).

Requisitos documentados em formato de histórias de usuário compõem o conjunto de entradas, eles devem estar descritos individualmente em arquivos tipo texto simples (.txt) no idioma inglês.

O padrão de histórias de usuário apresentado por Bourque *et al.* (2014) foi definido como modelo de especificação a ser submetido como entrada, possibilitando a identificação das *strings* delimitadoras, facilitando a busca dos termos, conforme segue.

Modelo:

As a <role>,

I want <goal/desire>

so that <benefit>.

Exemplo:

As a ***seller,***

I want to ***list products***

so that ***I can choose items to sell.***

Em concordância com a Figura 1, o conjunto de requisitos passa por 4 etapas até ser priorizado, sendo:

a) Preprocessamento de texto;
b) Busca de classes candidatas;

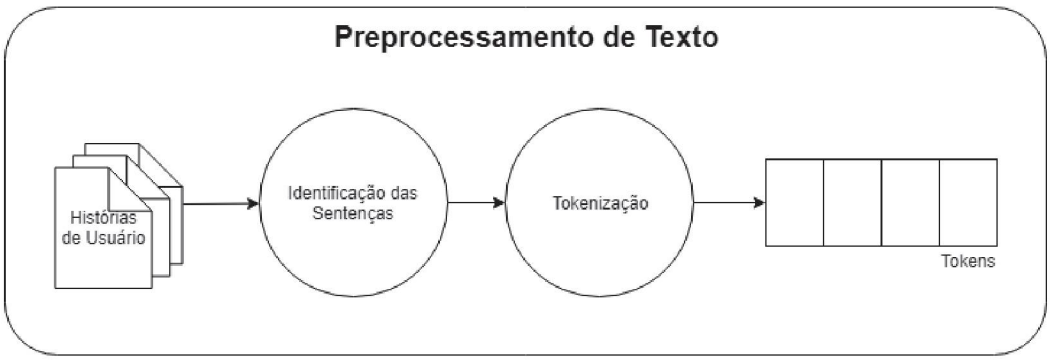
c) Detecção de dependências;

d) Priorização por *ranking* de dependências.

As 4 etapas são detalhadas nas próximas seções.

4.1 PREPROCESSAMENTO DE TEXTO

FIGURA 2: ETAPA 1: PREPROCESSAMENTO DE TEXTO.



FONTE: O autor (2018).

Conforme exposto na Figura 2, na etapa de preprocessamento de texto, os arquivos que contém os requisitos são recebidos, populando um *array* no qual cada posição representa uma história de usuário. A sentença identificada passa pelo processo de tokenização, criando um novo *array*, contendo os *tokens* reconhecidos. Para o exemplo citado na seção anterior (*As a seller, I want to list products so that I can choose items to sell*), o *array* gerado deve possuir 17 posições, conforme exemplificado na Figura 3.

FIGURA 3: REPRESENTAÇÃO DO *ARRAY* DE *TOKENS* PARA O EXEMPLO.

tokens:

As	a	seller	,	I	want	to	list	products	so	that	I	can	choose	itens	to	sell
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

FONTE: O autor (2018).

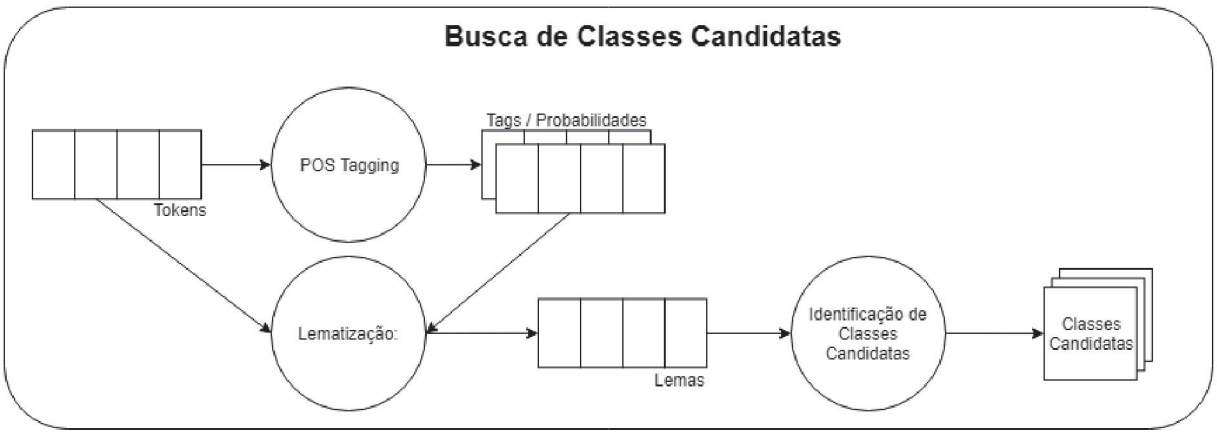
Com a conclusão da tokenização, o bloco de texto puro e bem definido, citado como objetivo do preprocessamento de texto segundo Indurkhya e Damerau (2010, p. 10), é obtido. Partindo dele, o método busca identificar as classes candidatas contidas nos textos que especificam requisitos para consecutivo mapeamento de dependências, essa etapa é detalhada na próxima seção.

4.2 BUSCA DE CLASSES CANDIDATAS V. 1.0.

Para o método, são consideradas classes candidatas os substantivos existentes nos requisitos do sistema em análise, conforme indicado por Paula Filho

(2003, p. 139). Para isso, o *array* de *tokens* gerado no preprocessamento é fornecido como entrada do processo de marcação de parte do discurso, o POS *Tagging* que, após a execução, cria outros dois *arrays*: *tags* e probabilidades. A Figura 4 exibe os elementos e passos necessários até a identificação dessas classes, que conclui a etapa.

FIGURA 4: ETAPA 2: BUSCA DE CLASSES CANDIDATAS.



FONTE: O autor (2018).

O *array* de *tags* recebe uma etiqueta com a descrição do papel do *token* da mesma posição na sentença. Por sua vez, o *array* de probabilidades vincula a probabilidade da atribuição da *tag* estar correta para o mesmo *token*. No exemplo, a configuração de *tags* esperada, considerando a lista apresentada por Schweinberger (2016), exibida no Quadro 5, Seção 2.5.2, pode ser visualizada na Figura 5.

FIGURA 5: REPRESENTAÇÃO DO *ARRAY* DE *TAGS* PARA O EXEMPLO.

tags:

IN	DT	NN	,	PRP	VBP	TO	VB	NNS	RB	IN	PRP	MD	VBD	NNS	TO	VB
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

FONTE: O autor (2018).

Assim sendo, após análise do mesmo *framework*, o *array* de probabilidades do exemplo deve ter a mesma quantidade de posições (17), populadas com valores percentuais, conforme mostra a Figura 6.

FIGURA 6: REPRESENTAÇÃO DO *ARRAY* DE PROBABILIDADES PARA O EXEMPLO.

probabilidades:

0.972	0.984	0.987	0.977	0.984	0.995	0.966	0.795	0.958	0.625	0.974	0.995	0.990	0.148	0.814	0.992	0.978
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

FONTE: O autor (2018).

Diante do exposto, pode-se observar que o *token* da posição 2, "*seller*", foi classificado com a *tag* NN, que indica um substantivo no singular e teve a probabilidade de 98,7% vinculada a essa atribuição. Por outro lado, o *token* da posição 13, "*choose*", classificado como VBD, ou seja, verbo no passado, recebeu a menor probabilidade, sendo 14,8%.

No próximo passo são definidos os lemas, agrupando os *tokens* identificados nas histórias de usuário em termos mais genéricos, evitando algumas ambiguidades. Para isso, os três *arrays* obtidos anteriormente (*tokens*, *tags* e probabilidades) devem ser analisados. Assim, obtém-se como saída um novo *array* contendo os lemas de cada história de usuário, em substituição ao de *tokens*, o resultado esperado é exemplificado na Figura 7.

FIGURA 7: REPRESENTAÇÃO DO *ARRAY* DE LEMAS PARA O EXEMPLO.

lemas:

As	a	seller	,	I	want	to	list	product	so	that	I	can	choose	item	to	sell
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

FONTE: O autor (2018).

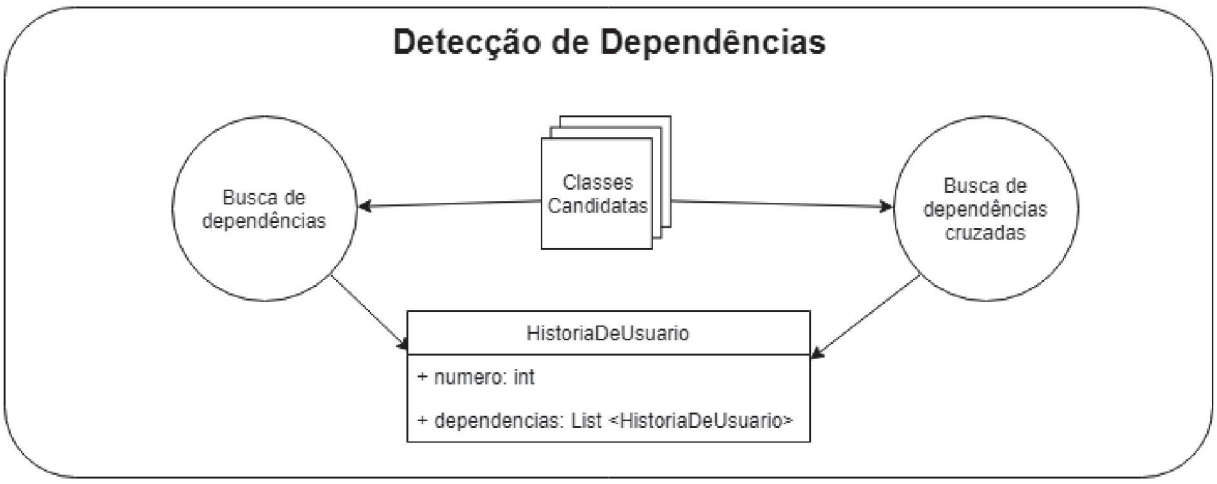
Diante disso, constata-se que as posições 8 e 14 foram alteradas em relação ao *array* de *tokens*. Isso indica que o *token "products"* corresponde ao lema *"product"*, enquanto o *token "itens"* corresponde ao lema *"item"*. Conforme averiguado na Seção 2.5.3, essa fase do processamento evita que palavras com a mesma raiz que apresentam formas de escrita distintas, sejam interpretadas como termos diferentes.

Finalmente, busca-se nos *arrays* de lemas, *tags* e probabilidades os substantivos (compostos ou não) mais relevantes classificando-os como classes candidatas do problema.

4.3 DETECÇÃO DE DEPENDÊNCIAS

As classes candidatas direcionam o processo de detecção de dependências já que conseguem revelar quais requisitos as manipulam. Portanto, essa etapa analisa as estruturas de dados que armazenam as classes encontradas e os requisitos que as controlam para realizar tal identificação. Esse processo é mostrado na Figura 8.

FIGURA 8: ETAPA 3: DETECÇÃO DE DEPENDÊNCIAS



FONTE: O autor (2018).

Ao encontrar uma classe candidata que é manipulada por 2 ou mais requisitos distintos, um *link* de dependência é estabelecido entre eles, caracterizando uma relação do tipo *Structural/Similar_to* (DAHLSTEDT; PERSSON, 2003) ou *Constraint/Requiring* (ZHANG, 2013) sem especificação de precedência, ou seja, os requisitos envolvidos apontam um para o outro.

Uma rotina auxiliar identifica dependências cruzadas de forma recursiva, buscando nas outras classes candidatas manipuladas pelas histórias de usuário do conjunto explorado, os mesmos *links* citados anteriormente, incrementando a lista de dependências quando uma nova for detectada.

O processo pode ser melhor entendido mediante aplicação do método proposto no conjunto de requisitos a seguir. Na notação, Rn corresponde ao identificador da história de usuário recebida, VP ao verbo principal encontrado, R ao substantivo do campo *role* (papel), G ao do campo *goal/desire* (objetivo) e B ao do campo *benefit* (benefício).

R1: *As a waiter I want to create a food order so that the kitchen can prepare it.*

VP: *Create.* **R:** *Waiter.* **G:** *Food_order* **B:** *Kitchen*

R2: *As a waiter I want to create a drink order so that the bar can prepare it.*

VP: *Create.* **R:** *Waiter.* **G:** *Drink_order.* **B:** *Bar.*

R3: *As a waiter I want to open the table bill so that I can add orders.*

VP: *Open.* **R:** *Waiter.* **G:** *Table_bill.* **B:** *Order.*

R4: *As a waiter I want to change an order so that I can fix a wrong order.*

VP: *Change.* **R:** *Waiter.* **G:** *Order.* **B:** *Order.*

R5: *As a waiter I want to cancel an order so that I can warn the kitchen or the bar.*

VP: *Cancel.* **R:** *Waiter.* **G:** *Order.* **B:** *bar.*

R6: *As a waiter I want to close the table bill so that the bill can be calculated.*

VP: *Close.* **R:** *Waiter.* **G:** *Table_bill.* **B:** *Bill.*

R7: *As a waiter I want to set the table so that I can open an empty bill.*

VP: *Set.* **R:** *Waiter.* **G:** *Table.* **B:** *Bill.*

R8: *As a waiter I want to print the bill so that I can deliver it to the customer.*

VP: *Print.* **R:** *Waiter.* **G:** *Table_bill.* **B:** *Customer.*

R9: *As a manager I want to view the list of orders so that I can distribute tasks.*

VP: *View.* **R:** *Manager.* **G:** *Order.* **B:** *Task.*

R10: *As a manager I want to manage the stock so that I can enable items on the menu.*

VP: *Manage.* **R:** *Manager.* **G:** *Stock.* **B:** *Menu.*

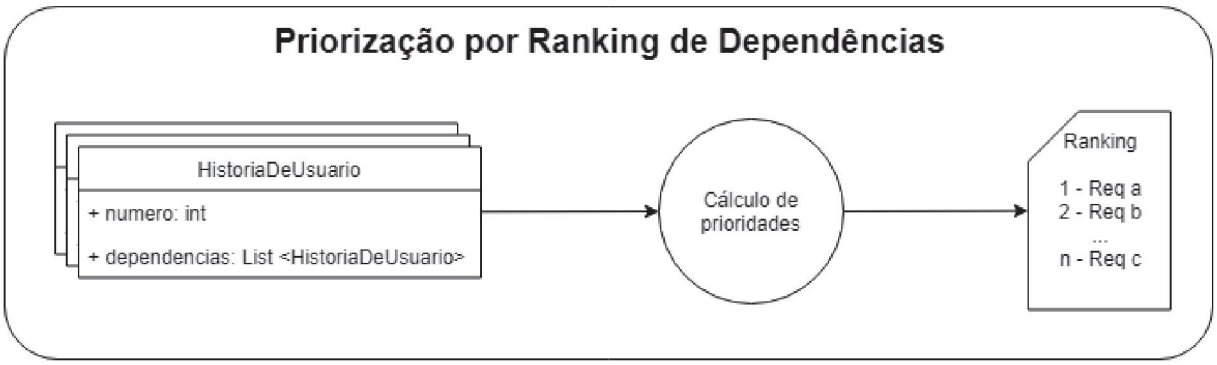
Os *links* são estabelecidos vinculando as histórias de usuário que referenciam uma mesma classe candidata. Por exemplo, cada requisito do conjunto [R3, R4, R5 e R9] receberá um *link* de dependência para os demais membros do conjunto por referenciarem a classe candidata *Order*. Adicionalmente, R3 receberá *link* para R6 e vice-versa por referenciarem a classe candidata *Table_bill*. Da mesma forma, R5 e R2 também serão vinculados pela referência a *Bar* e assim

sucessivamente, até que o método identifique todas as classes candidatas e dependências contidas no problema.

4.4 PRIORIZAÇÃO POR *RANKING* DE DEPENDÊNCIAS

O cálculo de prioridades é realizado ao receber o conjunto de histórias de usuário com suas respectivas estruturas de dados que contém as dependências identificadas na etapa anterior. As prioridades sugeridas são calculadas tendo em vista a quantidade de dependências de cada uma, ou seja, para o método, quanto mais *links* um requisito possui, maior é a prioridade sugerida para ele, esse processo é exemplificado na Figura 9.

FIGURA 9: ETAPA 4: PRIORIZAÇÃO POR DEPENDÊNCIAS.



FONTE: O autor (2018).

4.5 PROTÓTIPO DESENVOLVIDO

O protótipo que implementa o método, chamado de Dependency Rank, foi desenvolvido utilizando o Apache OpenNLP como *framework* de PLN e linguagem de programação Java. Em decorrência disso, os processos de tokenização, POS *tagging* e lematização foram implementados utilizando os métodos nativos do *framework*, assim como seu dicionário de sinônimos.

No preprocessamento de texto, o protótipo recebe os arquivos txt que contém as histórias de usuário instanciando um objeto que armazena a sentença e um número sequencial. Todo objeto criado executa o método de tokenização, aplicando o OpenNLP *tokenizer*, para popular seu *array* de *tokens*.

A etapa de busca de classes candidatas é realizada com suporte no *array* de *tokens* obtido. São aplicados os métodos POS *tagger* e *lemmatizer* nativos do Apache OpenNLP para obter os dados relativos a *tags*, probabilidades e lemas da sentença. Em seguida, são aplicados 4 métodos de identificação de elementos.

Inicialmente, um método busca o verbo principal do requisito, explorando o campo *goal/desire* da história de usuário, ou seja, o intervalo entre [..., *I, want, to*, ...] e [..., *so, that*, ...]. Nesse caso, o lema sinalizado como verbo, isto é, que contém "VB[6]" na *tag* associada, com maior probabilidade é retornado. Isso permite procurar substantivos ligados a esse lema e identificar a ação vinculada ao requisito.

Todos os lemas que potencialmente se refiram a verbos são avaliados considerando suas *tags*, probabilidades e a posição onde se encontram na história. O algoritmo exclui da avaliação os que não representam funcionalidades como *like*, *need*, *see, etc.* e retorna o de maior probabilidade dentro do intervalo.

Com o verbo principal identificado, o segundo método percorre o *array* de lemas entre as *strings* [*As, a, ...*] e [..., *I, want, to*, ...] em busca do *role* (papel) da história de usuário em análise. Ele verifica quais lemas constantes no intervalo são substantivos, ou seja, cuja posição associada no *array* de *tags* contém "NN[7]" em sua *string*. O substantivo com maior probabilidade vinculada é concatenado com substantivos vizinhos quando for o caso, sendo retornado como resultado desse processamento.

Para identificar a classe candidata do campo *goal/desire*, o terceiro método explora o intervalo entre as posições de [..., *I, want, to*, ...] e [..., *so, that*, ...] em busca do substantivo que melhor classifica esse campo da história. O substantivo de maior probabilidade entre os ligados ao verbo principal é concatenado com substantivos vizinhos, quando necessário, e retornado, sendo considerado a classe candidata do campo. O algoritmo exibido na Figura 10 mostra como os lemas são encontrados e concatenados.

---

6 Todas e apenas as *tags* que referem-se a verbos no *framework* Apache OpenNLP possuem VB em sua *string* conforme exibido pela Tabela 5.

7 Todas e apenas as *tags* que referem-se a substantivos no *framework* Apache OpenNLP possuem NN em sua *string* conforme exibido pela Tabela 5.

FIGURA 10: ALGORITMO DE BUSCA NO CAMPO *GOAL*

```
INTEIRO: pos = 0, posIWantTo = 0, posSoThat = 0, ind = 1, i
CARACTERE: classeCand
LOGICO: continua = V
REAL: prob = 0
INICIO
    posIWantTo = retornaPosIWantTo(sentenca)
    posSoThat = retornaPosSoThat(sentenca)
    PARA (i DE posIWantTo ATE posSoThat)
        SE (tags[i].contem("NN"))
            SE (prob < probs[i])
                pos = i
                classeCand = lemas[i]
            FIM-SE
        FIM-SE
    FIM-PARA

    ENQUANTO (continua)
        SE (tags[pos-ind].contem("NN"))
            classeCand = lemas[pos-ind] + "_" + classeCand
        SENAO
            continua = F
        FIM-SE
    FIM-ENQUANTO
    continua = V
    ENQUANTO (continua)
        SE (tags[pos+ind].contem("NN"))
            classeCand = classeCand + "_" + lemas[pos+ind]
        SENAO
            continua = F
        FIM-SE
    FIM-ENQUANTO
FIM
```

FONTE: O autor (2018)

Por fim, o quarto método busca após as posições de [..., *so*, *that*, ...] o substantivo de maior probabilidade que possa caracterizar uma classe candidata no campo *benefit*. Assim como nos demais casos, esse termo pode ser concatenado com as posições vizinhas do *array* de lemas caso sejam outros substantivos.

Tendo os 4 itens devidamente reconhecidos, uma rotina adiciona as três[8] classes candidatas a uma lista específica, identificando os números das histórias de usuário de onde foram extraídas. Cada classe só é adicionada uma vez, portanto, para cada ocorrência de classes já existentes, a relação de histórias de usuário que as manipulam é incrementada com o número da nova história envolvida, possibilitando a realização da próxima etapa. A Figura 11 exemplifica os retornos obtidos no exemplo iniciado na Seção 4.1.

8 Rotina alterada na segunda versão do método. A alteração é justificada e detalhada na Seção 4.5.1.

FIGURA 11: VERBO PRINCIPAL E CLASSES CANDIDATAS IDENTIFICADAS.

list	seller	product	item
verbo principal	classe candidata (role)	classe candidata (goal)	classe candidata (benefit)

FONTE: O autor (2018).

Para a detecção de dependências detalhada Seção 4.3, o algoritmo foca nas classes candidatas identificadas nos campos *goal* e *benefit.* Esse critério foi adotado para que os papéis (ou atores) do sistema não gerem dependências, caso contrário, os requisitos de atores com mais funcionalidades poderiam herdar muitas dependências que priorizariam esses requisitos em relação aos de outros atores com menos funcionalidades vinculadas.

4.5.1 Versão 1.1 da busca de classes candidatas

Após a realização dos dois experimentos detalhados adiante, no Capítulo 5, foi observada uma potencial melhoria nos resultados por meio da alteração da rotina de busca de classes candidatas. Tal alteração foi testada nos mesmos conjuntos de requisitos dos experimentos realizados, tendo seus resultados investigados separadamente.

A atualização modifica o tratamento empregado aos substantivos compostos na implementação inicial. Os algoritmos então desenvolvidos, concatenavam esses termos gerando uma classe candidata única. Porém, ao levantar a hipótese de que os termos podem representar conceitos independentes, que possuem relação entre si no contexto explorado, optou-se por alterar a implementação de modo que, além do termo concatenado, os substantivos simples também sejam considerados classes candidatas.

Como exemplo, considerando o requisito "*As a waiter I want to create a* ***food order*** *so that the kitchen can prepare it*", temos que, seu processamento pela versão 1.0, identifica no campo *goal*, apenas a classe candidata *Food_order*. Em contrapartida, seu processamento pela versão 1.1 identifica três classes: *Food*, *Order* e *Food_order*. Pode-se observar que, para o contexto de um restaurante, os termos *food* e *order* podem representar objetos independentes, validando a hipótese citada.

Entende-se que pode haver melhora nos resultados dependendo da frequência com que esses termos ratifiquem tal hipótese, pois, a partir disso, torna-

se possível identificar outros *links* entre requisitos anteriormente ausentes. No exemplo dado, a possibilidade de identificar *links* de dependência é aumentada pela inclusão dos dois termos não abrangidos na versão 1.0.

Isso significa que um hipotético requisito que descreva um pedido de bebidas (*drink order*) no mesmo sistema, pode gerar um *link* de dependência para o requisito descrito no exemplo pela abrangência ao termo *order*, situação que não ocorria na versão anterior. Esse tipo de relação de dependência é classificado como *Structural/Similar_to* (DAHLSTEDT; PERSSON, 2003), representando requisitos com características semelhantes.

4.6 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou a arquitetura do método proposto e as principais rotinas implementadas no protótipo desenvolvido. As 4 etapas de processamento foram discutidas citando elementos de entrada, rotinas de implementação e saídas desejadas, considerando um exemplo de história de usuário.

A etapa de busca de classes candidatas, apresentada na Seção 4.2, foi implementada em duas versões distintas pois, durante a realização dos experimentos uma hipotética possibilidade de melhorar os resultados foi observada. Essa alteração é justificada na Seção 4.5.1.

O protótipo que implementa o método proposto foi utilizado na realização dos experimentos que são foco do Capítulo 5. A contextualização desses experimentos assim como os resultados obtidos são discutidos a seguir.

5 EXPERIMENTOS

No intuito de validar o método proposto foram realizados 2 experimentos. O primeiro simula a utilização do método em um projeto de *software* real já priorizado e implementado, comparando o *ranking* sugerido pelo Dependency Rank com a sequência de implementação realizada pela equipe envolvida.

O segundo fornece um sistema hipotético como problema a ser priorizado para um grupo de desenvolvedores, comparando os resultados sobre identificação de classes candidatas e sequência de priorização sugerida pelo Dependency Rank com as avaliações dos participantes.

Em ambos os casos os resultados obtidos pelas duas versões do protótipo, discutidas na Seção 4.5 e subseções, são confrontados, possibilitando examinar o impacto causado pela alteração na rotina de busca de classes candidatas.

Com estes experimentos, espera-se que o método classifique os requisitos por prioridades produzindo resultados próximos aos indicados pelos desenvolvedores nos cenários investigados. Espera-se ainda que as classes candidatas compreendidas nos problemas submetidos sejam identificadas, gerando corretamente os *links* de dependência.

Os experimentos e resultados mencionados são apresentados nas próximas seções.

5.1 EXPERIMENTO 1: SISTEMA REAL JÁ PRIORIZADO

Uma relação de 14 requisitos, descritos em histórias de usuário, referentes a um sistema real denominado **SIGEOS: Sistema Integrado de Gerenciamento de Estoque e Ordem de Serviço**, foi submetido como entrada para a realização do experimento. As histórias de usuário foram traduzidas para o idioma inglês e adaptadas ao padrão de entrada utilizado pelo Dependency Rank.

O profissional responsável pela equipe que desenvolveu o sistema forneceu a lista de requisitos e a sequência de implementação executada, entendida como a ordem de priorização definida pela equipe. O julgamento de prioridades da equipe foi subjetivo, tendo como "principal critério de priorização o maior ganho de valor ao cliente" (NOGUEIRA, 2018). Sendo assim, não há como precisar quais fatores foram considerados por esses profissionais, ou seja, os critérios adotados pelas duas abordagens, quais sejam, o método e a equipe, são diferentes.

O conjunto de requisitos foi processado duas vezes pelo protótipo que implementa o método, sendo uma vez com a versão 1.0 e outra com a versão 1.1 da rotina de busca de classes candidatas, gerando como saídas para cada uma delas, a lista de classes candidatas reconhecidas e uma sequência de priorização sugerida. As sequências obtidas automaticamente foram comparadas à sequência de implementação da equipe de desenvolvimento com o propósito de avaliar a proximidade entre as prioridades.

Com base na sequência sugerida pela versão 1.0, os requisitos foram divididos em 2 grupos, sendo de alta e de baixa prioridade, respeitando as situações de empate observadas após o processamento. Ou seja, para os 14 requisitos submetidos como entrada, os primeiros 7 itens foram classificados como sendo de alta prioridade, restando aos demais a inclusão no grupo de baixa prioridade.

Foi estabelecido como parâmetro de inclusão para os 2 grupos, que requisitos com o mesmo nível de prioridade não poderiam estar em grupos diferentes, já que não foram definidos critérios de desempate. Com a adoção de tal parâmetro, seria possível ter quantidades diferentes de elementos nos grupos.

O valor alvo para comparação foi definido com suporte na sequência de implementação da equipe, exibida adiante na Seção 5.1.1, dividindo os requisitos em outros dois grupos, de alta e baixa prioridade, compostos pelo mesmo número de requisitos constantes nos grupos extraídos da análise do método. Nesse cenário, a análise dos resultados pretendeu comparar os requisitos prioritários de acordo com o Dependency Rank com os da equipe, identificando quantitativamente os possíveis acertos do método.

Em uma segunda conjuntura, a análise teve caráter subjetivo e se baseou nos níveis de prioridade encontrados automaticamente pela versão 1.0. Nesse caso, o número de elementos nos grupos foi definido pelos requisitos que receberam o mesmo nível de prioridade, ou seja, como foram identificados 4 níveis diferentes de prioridades entre os requisitos, conforme exibido na Figura 12 da Seção 5.1.2, a sequência da equipe foi desmembrada em 4 grupos, sendo 1 para cada nível, com a mesma quantidade de elementos. Para avaliar a proximidade do método no julgamento de cada requisito, foi analisada a distância da classificação do resultado obtido após execução do protótipo com relação à sequência de implementação fornecida pelo desenvolvedor.

O resultado da versão 1.1 é investigado comparando a sequência sugerida à executada pela equipe de forma simplificada, visto que, contempla apenas a atualização da rotina de busca de classes candidatas. Por fim, os resultados atingidos pela versão são confrontados aos da versão anterior com o propósito de fornecer um panorama do impacto da alteração realizada.

5.1.1 Elementos de entrada

Os 14 requisitos quem compõem o SIGEOS e foram processados pelo Dependency Rank são apresentados a seguir:

R1. *As an inventory administrator, I want to register an entry so that I can record the addition of a material to the stock.*

R2. *As an inventory administrator, I want to record an output so that I can record the use of a stock material.*

R3. *As an inventory administrator, I want to register a material so that I can manage its entries and outputs.*

R4. *As an inventory administrator, I want to list materials in the stock so that I can see the current amount of each material.*

R5. *As a stock administrator, I want to register a monthly closing so that I can calculate the output cost of each stock material.*

R6. *As a manager, I want to register a service order so that I can record building maintenance needs.*

R7. *As a manager, I want to include a follow-up for the service order so that I can register the responsibles for its execution and the time spent by each one.*

R8. *As a collaborator, I want to register a service order so that I can record the building maintenance needs.*

R9. *As an inventory administrator, I want to view materials under the minimum amount so that I can order new amounts when needed.*

R10. *As an inventory administrator, I want to enable a service so that I can hide from the requestor the services that are no longer available.*

R11. *As a manager, I want to view only the active services on the service order so that the inactive services won't be available.*

R12. *As a stock administrator, I want to block the input of duplicated products for the same commitment so that duplication cannot be made.*

R13. *As a manager, I want to view the ratio of materials and labor spent by a service order so that I can be able to know the total amount spent on each one.*

R14. *As a manager, I want to associate the service order with the requestor's department, so that reports can be issued on the costs of the it by each department.*

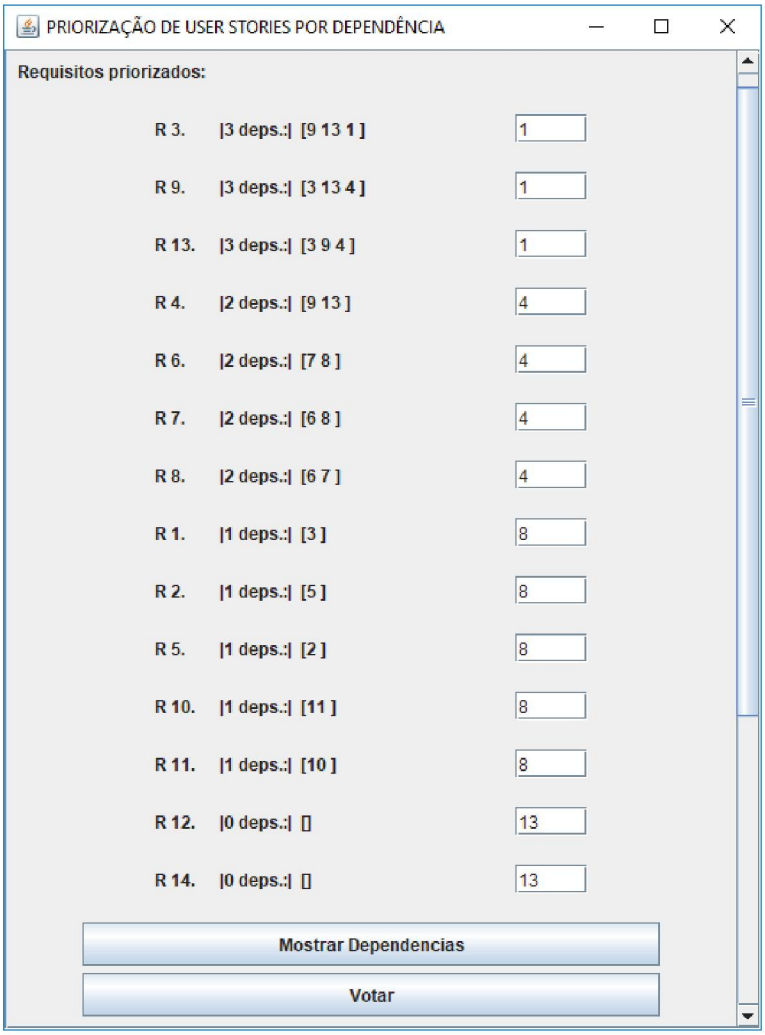
A ordem de implementação executada pela equipe de desenvolvimento é exibida a seguir:

1. R8.
2. R6.
3. R13.
4. R5.
5. R9.
6. R2.
7. R1.
8. R12.
9. R7.
10. R14.
11. R4.
12. R3.
13. R11.
14. R10.

5.1.2 Resultados da versão 1.0 do método no experimento

Após ser realizado o processamento do conjunto de requisitos pela versão 1.0 do Dependency Rank, ou seja, com a implementação da detecção de classes candidatas conforme descrito na Seção 4.2, o método identificou 4 grupos de prioridades entre os 14 requisitos fornecidos, fato que se deve às situações de empate nos números de dependências encontradas.

FIGURA 12: *RANKING* DE PRIORIDADES GERADO PELO MÉTODO.



FONTE: O autor (2018).

De acordo com o resultado computado pelo protótipo, os requisitos com maior nível de prioridade são R3, R9 e R13, com 3 dependências cada. O segundo grupo é composto por R4, R6, R7 e R8 com 2 dependências. No terceiro grupo foram classificados R1, R2, R5, R10 e R11 com 1 dependência encontrada. Finalmente, os requisitos R12 e R14, que não possuem dependências, foram considerados os menos prioritários.

No Quadro 9 são apresentados os dados que basearam a decisão. A coluna Req. mostra o identificador do requisito referido, V. P. o verbo principal encontrado, C. C. 1 (*role*) a classe candidata encontrada no campo *role* da história de usuário, C. C. 2 (*goal*) a classe candidata do campo goal, C. C. 3 (*benefit*) a classe candidata do campo *benefit* e, por fim, Deps. lista os *links* de dependência encontrados.

QUADRO 9: CLASSES CANDIDATAS E DEPENDÊNCIAS ENCONTRADAS.

Req.	V. P.	C. C. 1 (*role*)	C. C. 2 (*goal*)	C. C. 3 (*benefit*)	Deps.
R1	*Register*	*Inventory_administrator*	*Entry*	*Addition*	R3
R2	*Record*	*Inventory_administrator*	*Output*	*Stock_material*	R5
R3	*Register*	*Inventory_administrator*	*Material*	*Entry*	R1, R9, R13
R4	*List*	*Inventory_administrator*	*Stock*	*Amount*	R9, R13
R5	*Register*	*Stock_administrator*	*Closing*	*Stock_material*	R2
R6	*Register*	*Manager*	*Service_order*	*Building_maintenance_need*	R7, R8
R7	*Include*	*Manager*	*Service_order*	*Execution*	R6, R8
R8	*Register*	*Collaborator*	*Service_order*	*Building_maintenance_need*	R6, R7
R9	*View*	*Inventory_administrator*	*Material*	*Amount*	R3, R4, R13
R10	*Enable*	*Inventory_administrator*	*Service*	*Service*	R11
R11	*View*	*Manager*	*Service*	*Service*	R10
R12	*Block*	*Stock_administrator*	*Product*	*Duplication*	-
R13	*View*	*Manager*	*Material*	*Amount*	R3, R4, R9
R14	*Associate*	*Manager*	*Department*	*Department*	-

FONTE: O autor (2018).

No total foram identificadas 4 classes candidatas no campo *role*: *Inventory_administrator*, *Stock_administrator*, *Manager* e *Collaborator*. Para o Dependency Rank, as classes desse tipo não caracterizam dependências uma vez que representam apenas os atores do sistema. Adicioná-las à lista de dependências aumentaria a prioridade de requisitos potencialmente menos relevantes apenas pelo fato de serem solicitados para o mesmo tipo de usuário.

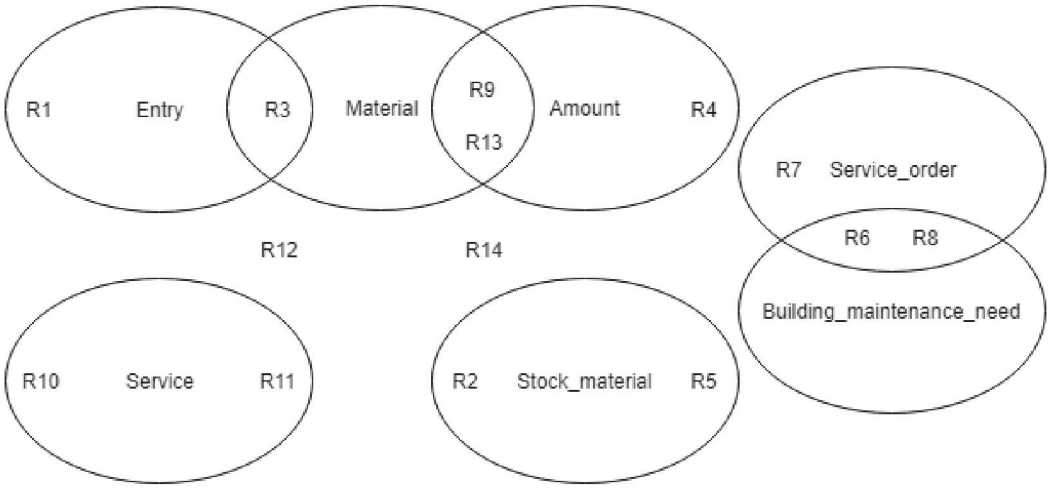
O método identificou também 9 classes candidatas no campo *goal*: *Entry*, *Output*, *Material*, *Stock*, *Closing*, *Service_order*, *Service*, *Product* e *Department*. Essas, assim como as do campo *benefit*, são responsáveis por gerar os *links* de dependência entre os requisitos. Portanto, a correta identificação dessas classes é determinante para o processo de priorização. A diferenciação dos termos *Service* e *Service_order* foi possível pela estratégia de concatenação de *strings* realizada após a lematização, tornando o termo composto independente do termo simples.

No campo *benefit*, foram identificadas outras 9 classes: *Addition*, *Stock_material*, *Entry*, *Amount*, *Building_maintenance_need*, *Execution*, *Service*,

Duplication e *Department*. Dentre essas, 7 foram identificadas apenas nesse campo enquanto 2, *Entry* e *Service* já haviam sido reconhecidas no campo *goal*.

Fundamentado nos resultados do Dependency Rank, observa-se que dos 14 requisitos que compõem o conjunto de entrada, apenas 2 não possuem *links* de dependências: R12 e R14. Pode-se notar na Figura 13, a influência das classes candidatas dos campos *goal* e *benefit* na geração dos *links* para os outros casos.

FIGURA 13: CONJUNTOS DE REQUISITOS MANIPULADOS PELAS CLASSES CANDIDATAS DO EXPERIMENTO.



FONTE: O autor (2018).

Entre os requisitos classificados com maior nível de prioridade, R3 possui dependências com R9 e R13 e vice-versa pela referência a *Material*, e a R1 pela referência a *Entry*. R9 e R13 dependem, além de R3, já citado, de R4 por referenciarem *Amount*.

No segundo nível, com duas dependências encontradas, R4 possui *link* com R9 e R13 por referência à classe candidata *Amount*. R6, R7 e R8 possuem *link* entre si por referenciarem *Service_order.* Além disso, R6 e R8 também são interdependentes pela referência a *Building_maintenance_need*.

No grupo de requisitos com apenas uma dependência identificada, R1 possui *link* com R3 pela referência a *Entry*, R2 e R5 são dependentes entre si pela referência a *Stock_material*, assim como R10 e R11 pela referência a *Service*.

5.1.2.1 Comparação entre grupos de alta e baixa prioridade

De posse dos resultados gerados, o grupo de alta prioridade foi composto pelos requisitos com 2 e 3 dependências, conforme análise do método.

Considerando que os 7 requisitos com maior nível de prioridade de acordo com o método atendem a esse critério, eles foram comparados aos 7 de maior prioridade, constantes na ordem de implementação da equipe de desenvolvimento.

Sendo assim, os requisitos com 1 ou nenhuma dependência de acordo com o Dependency Rank formam o grupo de baixa prioridade. Portanto, os 7 requisitos desse grupo foram comparados aos 7 de menor prioridade da lista da equipe. As Tabelas 2 e 3 exibem os resultados obtidos nestes 2 grupos. Aparecem em destaque os acertos do método.

TABELA 2: COMPARAÇÃO ENTRE OS REQUISITOS DE ALTA PRIORIDADE.

Prioridade Dependency Rank	Requisito	Prioridade Equipe
1 (alta)	R3	12 (baixa)
4 (alta)	R4	11 (baixa)
4 (alta)	**R6**	**2 (alta)**
4 (alta)	R7	9 (baixa)
4 (alta)	**R8**	**1 (alta)**
1 (alta)	**R9**	**5 (alta)**
1 (alta)	**R13**	**3 (alta)**

FONTE: O autor (2018).

Considerando o grupo dos 7 requisitos de maior prioridade para a equipe, o Dependency Rank classificou corretamente 4 itens, fato que pode ser observado pelos requisitos R6, R8, R9 e R13 que encontram-se nas primeiras posições de ambas as listas. R6 e R8 se referem ao cadastro inicial de uma ordem de serviço de manutenção predial, possuindo apenas atores diferentes. Neste ínterim, R9 contempla a gestão dos materiais que estão abaixo do estoque mínimo desejado, possibilitando um novo pedido. Já R13 tem por finalidade calcular o valor gasto entre materiais e mão de obra envolvidos em uma ordem de serviço.

Os requisitos R3, R4 e R7 não constam entre os prioritários da equipe, porém, o Dependency Rank os julgou com alto nível de prioridade. O requisito R3 se refere ao cadastro inicial de materiais. R4 descreve a necessidade de listar a quantidade de cada material no estoque, enquanto R7 descreve o acompanhamento das ordens de serviço.

A Tabela 3 compara os resultados do método em relação aos requisitos classificados como de baixa prioridade:

TABELA 3: COMPARAÇÃO ENTRE OS REQUISITOS DE BAIXA PRIORIDADE.

Posição Método	Requisito	Posição Equipe
8 (baixa)	R1	7
8 (baixa)	R2	6
8 (baixa)	R5	4
8 (baixa)	**R10**	**14**
8 (baixa)	**R11**	**13**
13 (baixa)	**R12**	**8**
13 (baixa)	**R14**	**10**

FONTE: O autor (2018).

Assim como ocorreu com o grupo de alta prioridade, novamente o método identificou corretamente 4 entre 7 itens, sendo R10, R11, R12 e R14. Desses, R12 e R14 não possuem dependências vinculadas, como manipulam classes candidatas que não foram encontradas em outras histórias de usuário, obtiveram o menor nível de prioridade do problema.

A funcionalidade representada por R10 é a habilitação de serviços para novas requisições. R11 trata da visualização dos serviços ativos pelo gerente. R12 descreve a necessidade de bloquear entradas duplicadas para os produtos. Por fim, R14 solicita a associação de uma ordem de serviço a um departamento da empresa.

No grupo de baixa prioridade da equipe não constam os requisitos R1, R2 e R5, no entanto, para esse 3 itens o método encontrou apenas 1 dependência. Os requisitos R1 e R2 são complementares e descrevem a necessidade de registrar, respectivamente, entradas e saídas de materiais. Enquanto isso, R5 solicita a realização de um fechamento mensal para os materiais do estoque.

É possível perceber que a forma de escrever as histórias de usuário influencia diretamente nos resultados obtidos. Utilizando R2 como exemplo, temos que a classe candidata encontrada no campo *goal* foi *Output*, por ser o substantivo mais relevante pelo julgamento do Dependency Rank. Porém, ao analisar a sentença, percebe-se que a saída (*output*) se refere aos materiais do estoque.

Nesse caso, a utilização da palavra *material* no campo *goal* poderia melhorar sensivelmente a classificação de prioridade de R2, desde que fosse corretamente interpretado pelos algoritmos do método. Na hipótese, a adaptação acarretaria em *links* para R3, R9 e R13. Assim, R2 passaria a ter 4 dependências ante 1 identificada sem a alteração, tendo sua posição sugerida priorizada para o maior nível do

problema. Logo, R2 poderia ser classificado junto de R3, R9 e R13, que teriam sua quantidade de dependências incrementada em uma. A situação é ilustrada adiante:

R2 (alterado): *As an inventory administrator, I want to record an output* ***of a material*** *so that I can record the use of a stock material.*

O trecho em destaque (*of a material*) foi adicionado posteriormente. Mesmo diante dessa possibilidade, vale destacar que o escopo do Dependency Rank visa trabalhar com requisitos descritos em texto livre, padronizando as entradas o mínimo possível, realizando a busca por dependências e posterior sugestão de priorização para análise do usuário do protótipo.

5.1.2.2 Comparação considerando os grupos identificados pelo método

Conforme apresentado nas seções anteriores, a execução do protótipo para o exemplo identificou 4 níveis diferentes de prioridades. Apoiado nesse resultado, os requisitos classificados pelo método automatizado foram separados em 4 grupos.

Os 3 requisitos com maior nível de prioridade (R3, R9 e R13) foram selecionados como os componentes do grupo 1. Os 4 do segundo nível (R4, R6, R7 e R8) foram selecionados para compor o grupo 2. Os 5 do terceiro nível (R1, R2, R5, R10 e R11) compuseram o grupo 3, enquanto os 2 requisitos com menor nível de prioridade (R12 e R14), nos quais não foram encontradas dependências, compuseram o grupo 4, conforme Quadro 10.

QUADRO 10: COMPOSIÇÃO DOS GRUPOS DO DEPENDENCY RANK.

Dependency Rank	Grupo 1	R3, R9, R13
	Grupo 2	R4, R6, R7, R8
	Grupo 3	R1, R2, R5, R10, R11
	Grupo 4	R12, R14

FONTE: O autor (2018).

O mesmo processo de divisão em grupos foi realizado considerando a sequência fornecida pela equipe de desenvolvimento, respeitando o tamanho de cada grupo anteriormente definido. Essa divisão é exibida no Quadro 11. Sendo assim, foram adicionados ao grupo 1 da equipe os requisitos R8, R6 e R13. O grupo 2 foi composto por R1, R2, R5 e R9. O grupo 3 por R3, R4, R7, R12 e R14. Por fim, ao grupo 4 da equipe foram adicionados R10 e R11.

QUADRO 11: COMPOSIÇÃO DOS GRUPOS DA SEQUÊNCIA DA EQUIPE.

Sequência da Equipe	Grupo 1	R8, R6, R13
	Grupo 2	R1, R2, R5, R9
	Grupo 3	R3, R4, R7, R12, R14
	Grupo 4	R10, R11

FONTE: O autor (2018).

A Tabela 4 apresenta a distribuição dos requisitos do problema entre os grupos nos dois cenários. A coluna "*Ranking* automatizado" exibe o grupo em que o requisito foi classificado pelo método, a coluna "*Ranking* da equipe" exibe o grupo em que o requisito foi classificado pela equipe de desenvolvimento, seguindo os critérios de classificação. A coluna "Variação" compara a posição em que o item foi classificado pelo método, tendo como valor de referência a posição no *ranking* da equipe. Valores positivos indicam que o item foi classificado em grupo de prioridade superior pelo Dependency Rank, se comparado ao extraído da equipe, enquanto o valor negativo indica que o método classificou com prioridade inferior na comparação.

TABELA 4: COMPARAÇÃO ENTRE OS 4 GRUPOS ENCONTRADOS.

Requisito	***Ranking* automatizado**	***Ranking* da equipe**	**Variação**
R1	3	2	-1
R2	3	2	-1
R3	1	3	+2
R4	2	3	+1
R5	3	2	-1
R6	2	1	-1
R7	2	3	+1
R8	2	1	-1
R9	1	2	+1
R10	3	4	+1
R11	3	4	+1
R12	4	3	-1
R13	1	1	0
R14	4	3	-1

FONTE: O autor (2018).

Entre os 14 requisitos avaliados, o método classificou com precisão apenas R13, no grupo de alta prioridade. Por outro lado, em apenas 1 caso, com R3, a classificação do método errou por mais de 1 grupo de diferença.

Nos demais 12 casos, ou seja, para 85% das situações contidas nesse problema, os requisitos foram classificados com um grupo de variação para mais ou para menos, tal fato sugere que mesmo as dependências encontradas automaticamente não fornecem resultados com alta precisão quando utilizadas como único critério de priorização, ainda assim, servem para obter resultados aproximados.

5.1.3 Resultados da versão 1.1 do método no experimento

O conjunto de requisitos fornecido pela equipe foi processado novamente, dessa vez abrangendo a alteração descrita na Seção 4.5.1. A alteração no algoritmo de classificação de classes candidatas ocasionou o efeito esperado, encontrando mais dependências e modificando o *ranking* da versão 1.0, conforme exposto em seguida.

A Tabela 5 possibilita visualizar a proximidade atingida pela versão 1.1 com relação às prioridades da equipe. Observando as situações de empate nos resultados, a coluna "*Ranking* v. 1.1" exibe o intervalo possível para o requisito após análise do método. A coluna "Variação mínima" mostra a quantidade mínima de posições que o requisito variou ao contrapor os dois *rankings*, facilitando a interpretação.

TABELA 5: RESULTADO DA VERSÃO 1.1.

Requisito	***Ranking* v. 1.1**	***Ranking* da equipe**	**Variação mínima**
R1	12	7	5
R2	Entre 1 e 5	6	1
R3	Entre 1 e 5	12	7
R4	Entre 6 e 10	11	1
R5	Entre 1 e 5	4	0
R6	Entre 6 e 10	2	4
R7	11	9	2
R8	Entre 6 e 10	1	5
R9	Entre 1 e 5	5	0
R10	Entre 6 e 10	14	4
R11	Entre 6 e 10	13	3

(continuação)

Requisito	*Ranking* v. 1.1	*Ranking* da equipe	Variação mínima
R12	Entre 13 e 14	8	5
R13	Entre 1 e 5	3	0
R14	Entre 13 e 14	10	3

FONTE: O autor (2018).

Nesse enquadramento, 3 requisitos são classificados virtualmente na posição correta, R5, R9 e R13, que apresentaram variação mínima de 0. Outros 2 atingiram uma posição de variação mínima, casos de R2 e R4.

Para 3 requisitos, quais sejam, R7, R11 e R14, as variações mínimas não ficaram muito distantes do valor alvo levando em conta o tamanho e a heterogeneidade do conjunto. R7 atingiu variação mínima de 2, representando 14,28% de distância ao valor alvo, enquanto para R11 e R14 foi atribuído o valor 3, representando distância de 21,43% do valor alvo.

Os demais requisitos foram classificados com distância de, pelo menos, 4 posições, sendo: R1, R3, R6, R8, R10 e R12.

5.1.4 Comparativo de proximidades entre as versões

Devido à heterogeneidade de níveis de prioridade atingidos pelas duas versões, não foi possível compará-los em cenário equivalente ao apresentado na Seção 5.1.2.2, dificultando a análise sobre a melhoria do ranqueamento. Contudo, a Tabela 6 compara as versões 1.0 e 1.1 apresentando as alterações nas quantidades de dependências e posições no *ranking*.

TABELA 6: DIFERENÇAS ENTRE AS VERSÕES 1.0 E 1.1.

Requisito	Deps. V. 1.0	*Ranking* V. 1.0	Deps. V. 1.1	*Ranking* V. 1.1
R1	1	8	1	12
R2	1	8	5	1
R3	3	1	5	1
R4	2	4	4	6
R5	1	8	5	1
R6	2	4	4	6
R7	2	4	2	11
R8	2	4	4	6
R9	3	1	5	1
R10	1	8	4	6
R11	1	8	4	6

(continuação)

Requisito	Deps. V. 1.0	*Ranking* V. 1.0	Deps. V. 1.1	*Ranking* V. 1.1
R12	0	13	0	13
R13	3	1	5	1
R14	0	13	0	13

FONTE: O autor (2018).

Conforme esperado, as posições anteriormente sugeridas foram alteradas em razão do novo critério de classificação de classes candidatas. Os requisitos com maior prioridade na avaliação da versão 1.0 possuíam 3 dependências, enquanto na versão 1.1 esse número foi elevado para 5. Os requisitos com maior alteração foram R2 e R5, nos quais foi identificada apenas 1 dependência na versão 1.0. Na nova implementação ambos atingiram o maior nível de prioridade, com 5 dependências.

Além de R2 e R5, o conjunto de dependências também variou entre outros requisitos, a saber:

- R3: de 3 para 5;
- R4: de 2 para 4;
- R6: de 2 para 4;
- R8: de 2 para 4;
- R9: de 3 para 5;
- R10: de 1 para 4;
- R11: de 1 para 4;
- R13: de 3 para 5.

A quantidade de dependências não foi alterada para R1, R7, R12 e R14.

Diante do exposto, fundamentando a análise nos valores de variação mínima e, observando que a versão 1.1 classificou corretamente 3 requisitos, atingindo baixa variação de posições para outros 5, presume-se que houve melhoria na ordem de priorização em relação à versão anterior.

5.2 EXPERIMENTO 2: VALIDAÇÃO POR PROFISSIONAIS

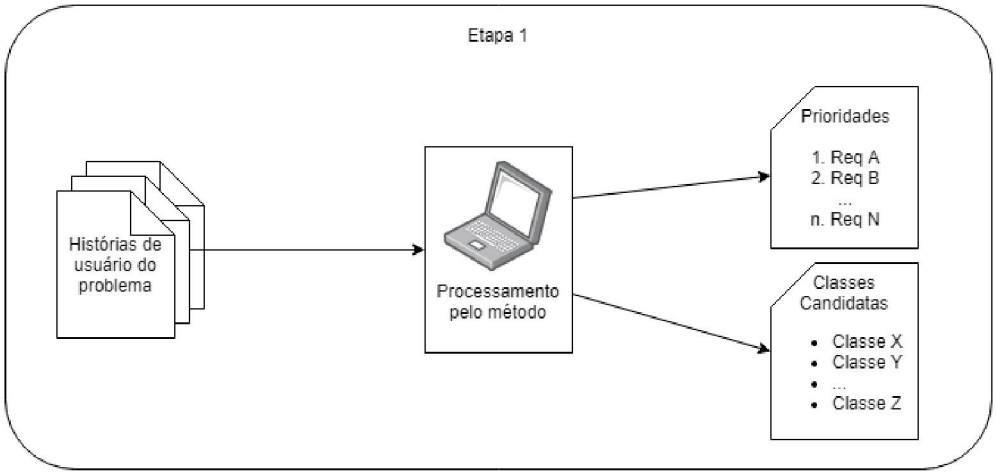
Um sistema hipotético com 10 requisitos especificados em histórias de usuário foi submetido à análise de 12 profissionais da área de tecnologia da informação com conhecimento acadêmico e/ou prático em priorização de requisitos de *software*. No experimento, os profissionais realizaram tarefas semelhantes às do

método proposto, identificando as classes candidatas envolvidas nas histórias de usuário e julgando as prioridades entre os requisitos. Os dados das avaliações individuais foram informados em um formulário formato *web*.

Os participantes responderam perguntas a respeito da experiência profissional, formação acadêmica, idade e questões técnicas, principalmente, sobre as classes candidatas envolvidas no problema e a sequência de priorização sugerida para o conjunto de requisitos fornecido. Diante da dificuldade de se obter um *ranking* geral a partir das avaliações, a mediana das posições foi utilizada para elaborar o valor alvo de priorização dos requisitos, possibilitando a confrontação com os resultados gerados pelo processamento automatizado.

O protótipo que implementa o método foi executado com as histórias de usuário submetidas como entrada por suas 2 versões. Os resultados referentes à identificação de classes candidatas e prioridades foram reservados para tal comparação, conforme ilustrado na Figura 14.

FIGURA 14: EXECUÇÃO DO PROTÓTIPO.



FONTE: O autor (2018).

O experimento foi realizado propondo atender dois objetivos:

a) Comparar o conjunto de classes candidatas reconhecidas pelo método com as classes candidatas citadas pelos profissionais;
b) Comparar o *ranking* de priorização sugerido pelo método com o extraído pelas avaliações dos profissionais.

Entendeu-se que a comparação entre as classes candidatas identificadas automaticamente e as citadas pelos participantes auxilia na interpretação dos

resultados, uma vez que, o correto reconhecimento dessas, influencia diretamente no ranqueamento implementado.

O *ranking* de prioridades informado pelos participantes foi definido subjetivamente de acordo com seus próprios critérios de avaliação, não foram indicadas quaisquer técnicas ou estipulados parâmetros para direcionar essa tarefa. Portanto, a comparação entre os dois *rankings* visa compreender o impacto das relações de dependências no julgamento subjetivo dos profissionais, bem como, se esse critério pode ser usado como determinante para priorizar os requisitos de *software* avaliados.

Similarmente ao realizado no experimento 1, da Seção 5.1 e subseções, os resultados alcançados pelas duas versões do método foram comparados aos valores alvo, buscando descobrir a melhor solução para o problema. Essa análise será apresentada adiante, nas Seções 5.2.7 para a versão 1.0 e 5.2.8 para a versão 1.1. Complementarmente, a Seção 5.2.9 confronta as saídas das versões discutindo as principais diferenças.

5.2.1 Elementos de entrada

O conjunto composto por dez requisitos analisados pelos profissionais é o mesmo do exemplo da Seção 4.3, conforme segue:

R1. As a waiter I want to create a food order so that the kitchen can prepare it.

R2. As a waiter I want to create a drink order so that the bar can prepare it.

R3. As a waiter I want to open the table bill so that I can add orders.

R4. As a waiter I want to change an order so that I can fix a wrong order.

R5. As a waiter I want to cancel an order so that I can warn the kitchen or the bar.

R6. As a waiter I want to close the table bill so that the bill can be calculated.

R7. As a waiter I want to set the table so that I can open an empty bill.

R8. As a waiter I want to print the bill so that I can deliver it to the customer.

R9. As a manager I want to view the list of orders so that I can distribute tasks.

R10. As a manager I want to manage the stock so that I can enable items in the menu.

Os resultados referentes a classes candidatas encontradas e *rankings* de priorização sugeridos pelas versões do protótipo são apresentados na sequência.

5.2.2 Resultados obtidos pela versão 1.0 do Dependency Rank

Após a execução do protótipo em sua versão 1.0, foram encontradas 14 classes candidatas nas histórias de usuário do conjunto submetido: *Waiter* e *Manager* no campo *role* das histórias de usuário; *Food_order*, *Drink_order*, *Table_bill*, *Order*, *Table e Stock* no campo *goal*; *Kitchen*, *Bar*, *Bill*, *Customer*, *Task* e *Menu* no campo *benefit* – onde também foi identificada *Order*, já apontada no campo *goal*.

O Quadro 12 apresenta a referida relação vinculando-as aos requisitos correspondentes.

QUADRO 12: CLASSES CANDIDATAS IDENTIFICADAS NA VERSÃO.

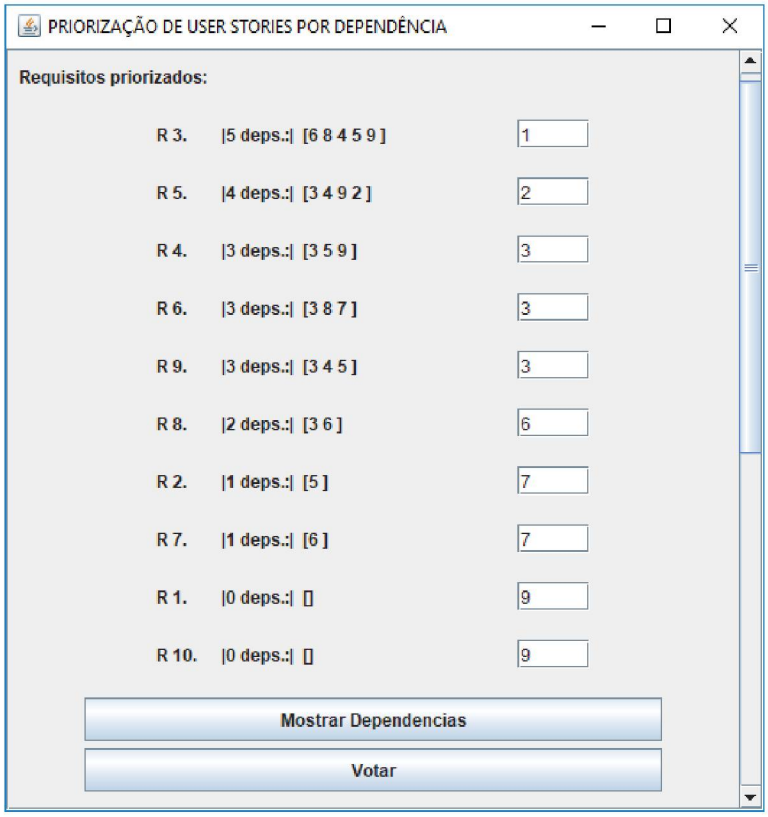
Req.	V. P.	C. C. 1 (*role*)	C. C. 2 (*goal*)	C. C. 3 (*benefit*)	Deps.
R1	*Create*	*Waiter*	*Food_order*	*Kitchen*	-
R2	*Create*	*Waiter*	*Drink_order*	*Bar*	R5
R3	*Open*	*Waiter*	*Table_bill*	*Order*	R4, R5, R6, R8, R9
R4	*Change*	*Waiter*	*Order*	*Order*	R3, R5, R9
R5	*Cancel*	*Waiter*	*Order*	*Bar*	R2, R3, R4, R9
R6	*Close*	*Waiter*	*Table_bill*	*Bill*	R3, R8, R7
R7	*Set*	*Waiter*	*Table*	*Bill*	R6
R8	*Print*	*Waiter*	*Table_bill*	*Customer*	R3, R6
R9	*View*	*Manager*	*Order*	*Task*	R3, R4, R5
R10	*Manage*	*Manager*	*Stock*	*Menu*	-

FONTE: O autor (2018).

Assim como anteriormente apresentado, no Quadro 9 da Seção 5.1.2, a coluna Req. mostra o identificador do requisito referido, V. P. o verbo principal encontrado, C. C. 1 (*role*) a classe candidata encontrada no campo *role* da história de usuário, C. C. 2 (*goal*) a classe candidata do campo goal, C. C. 3 (*benefit*) a classe candidata do campo *benefit* e, por fim, Deps. lista os *links* de dependência encontrados.

Diante disso, o *ranking* sugerido após execução do protótipo é exibido na Figura 15.

FIGURA 15: TELA DE SUGESTÃO DE PRIORIDADES DO PROTÓTIPO.



FONTE: O autor (2018).

A lista exibida é composta pelo identificador do requisito, a quantidade de dependências encontradas, a lista de requisitos dependentes e a posição sugerida para o ranqueamento. Exemplo: na Figura 15 o requisito R3 foi vinculado a cinco dependências (R6, R8, R4, R5 e R9), sendo sugerida a atribuição da primeira posição do *ranking*.

Por não possuir outros parâmetros de avaliação, situações de empate ocorrem com frequência, devendo ser resolvidas pelo usuário do protótipo. Porém, com a finalidade de auxiliar na tomada de decisão, a lista dos requisitos dependentes de cada uma das histórias de usuário listadas é exibida, facilitando as decisões que demandam interpretação humana.

No intuito de facilitar a análise, a ordem gerada pelo método foi comparada ao *ranking* extraído das avaliações dos profissionais respeitando as situações de empate constatadas. Os critérios adotados e a comparação com as avaliações dos profissionais são discutidos adiante, na Seção 5.2.7.

5.2.3 Resultados obtidos pela versão 1.1 do Dependency Rank

Com o processamento do conjunto de requisitos de entrada pela versão 1.1 do protótipo, o *ranking* previamente sugerido foi sensivelmente alterado. Isso se deve à estratégia de identificação de classes candidatas revelar itens até então ignorados em algumas histórias de usuário. Tal resultado pode ser melhor visualizado na Tabela 7.

Os números constantes em "Deps. V. 1.1" especificam a quantidade de dependências identificadas na versão. A coluna "*Ranking* V. 1.1" exibe a posição de priorização sugerida na versão citada. Os dados foram organizados respeitando a ordem do novo *ranking*.

TABELA 7: RESULTADO DA VERSÃO 1.1.

Requisito	**Deps. V. 1.1**	***Ranking* V. 1.1**
R3	8	1
R1	5	2
R2	5	2
R4	5	2
R5	5	2
R9	5	2
R6	3	7
R7	3	7
R8	3	7
R10	0	10

FONTE: O autor (2018).

Nesse enquadramento, as alterações mais significativas entre os números de dependência outrora reconhecidos ocorreram com R1 e R2, que receberam 5 dependências na nova versão. Na versão anterior R2 teve apenas uma dependência e R1 nenhuma.

Assim como ocorreu com o número de dependências, as posições dos requisitos no novo *ranking* também sofreram alterações. Também sob esse aspecto, R1 e R2 apresentaram as maiores alterações. Mesmo com as situações de empate observadas, a afirmação é prontamente validada ao notar que, enquanto R1 havia sido classificado com o último nível de prioridade pela versão anterior, dessa vez ficou atrás apenas de R3, empatado como 2º de maior prioridade junto a outros 4 requisitos. Entre eles está R2, anteriormente classificado em 7º entre 10 elementos.

Essas duas alterações foram viabilizadas pela ocorrência do termo *order* em R1 e R2 pois, com a nova estratégia, foi reconhecido como uma classe candidata separadamente dos termos *food* e *drink*.

A relação de classes candidatas reconhecidas pela nova versão foi composta por mais itens que a da versão 1.0. O Quadro 13 exibe a citada relação seguida pelos *links* de dependência detectados.

QUADRO 13: RELAÇÃO DE CLASSES CANDIDATAS E DEPENDÊNCIAS DA VERSÃO 1.1

Requisito	**V. P.**	**Classes Candidatas**	***Links* de dependência**
R1	*Create*	*waiter, food_order, food, order, kitchen*	R2, R3, R4, R5, R9
R2	*Create*	*waiter, drink_order, drink, order, bar*	R1, R3, R4, R5, R9
R3	*Open*	*waiter, table_bill, table, bill, order*	R1, R2, R4, R5, R6, R7, R8, R9
R4	*Change*	*waiter, order*	R1, R2, R3, R5, R9
R5	*Cancel*	*waiter, order, bar*	R1, R2, R3, R4, R9
R6	*Close*	*waiter, table_bill, table, bill*	R3, R7, R8
R7	*Set*	*waiter, table, bill*	R3, R6, R8
R8	*Print*	*waiter, table_bill, table, bill, customer*	R3, R6, R7
R9	*View*	*manager, order, task*	R1, R2, R3, R4, R5
R10	*Manage*	*manager, stock, menu*	-

FONTE: O autor (2018).

Adiante, na Seção 5.2.8, os resultados obtidos por essa versão são comparados às avaliações dos profissionais.

5.2.4 Questionário

O questionário elaborado foi formado por 4 questões relacionadas ao perfil dos sujeitos, 4 questões técnicas a respeito do problema dado – sendo 2 fechadas e 2 abertas – e a lista de requisitos (R1 – R10) fornecida para o experimento.

Questões técnicas:

Questão 1.1.: As histórias de usuário fornecidas representam possíveis requisitos de *software*?

Respostas: Sim, Não, Parcialmente (questão fechada).

Objetivo: Avaliar a validade das sentenças fornecidas enquanto requisitos de *software*, evitando que os resultados contemplem sentenças inválidas ou que caracterizem necessidades alheias à ES.

Questão 1.2.: O conjunto desses requisitos forma um sistema ou parte de um sistema?

Respostas: Sim, Não (questão fechada).

Objetivo: Investigar se os requisitos de fato formam um sistema de informação permitindo que os módulos se integrem em busca de atingir um objetivo em comum.

Questão 1.3.: Na sua opinião, quais termos representam classes candidatas no conjunto de requisitos do problema? Informe separando-as com ",".

Questão aberta.

Objetivo: Obter um conjunto de classes candidatas identificadas pelos profissionais, constituindo um suporte para comparação entre elas e o conjunto de classes reconhecidas pelo método.

Questão 1.4.: Com base nos requisitos fornecidos, qual seu *ranking* de prioridades? Informe em ordem decrescente utilizando o número das histórias do usuário.

Questão aberta.

Objetivo: Interpretar as prioridades informadas pelos profissionais possibilitando a comparação com o *ranking* atingido após execução dos algoritmos do método.

Questões sobre o perfil do profissional:

Questão 2.1.: Seu ano de nascimento (aaaa):

Questão aberta.

Questão 2.2.: Sexo:

Respostas: Masculino, Feminino (questão fechada).

Questão 2.3.: Formação na área de T.I. - informar nível e nome do(s) curso(s) ou “nenhum” caso não possua:

Questão aberta.

Questão 2.4.: Experiência em desenvolvimento de *software* (em anos):

Questão aberta.

5.2.5 Sujeitos

Foram sujeitos da pesquisa 12 profissionais da área de tecnologia da informação com formação acadêmica igual ou superior à graduação e conhecimentos em priorização de requisitos de *software*. Um panorama do perfil dos

participantes, baseado nas respostas informadas nas questões 2.1, 2.2, 2.3 e 2.4, é apresentado na Tabela 8.

TABELA 8: SUJEITOS POR ANO DE NASCIMENTO.

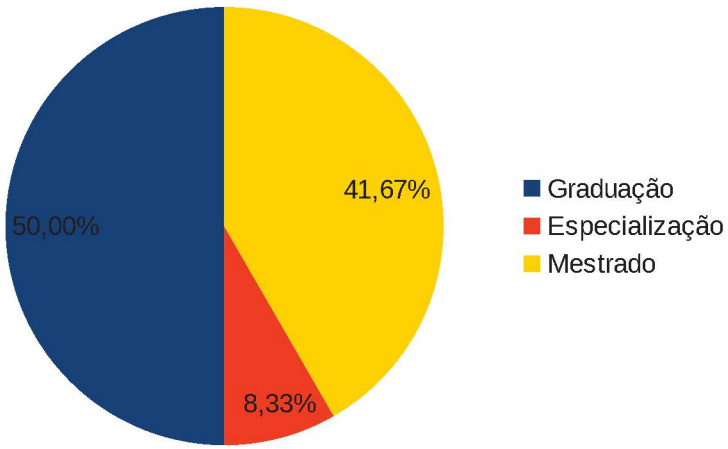
Ano de Nascimento	Quantidade
1971	2
1985	2
1986	1
1989	1
1991	1
1993	1
1994	1
1995	2
1996	1

FONTE: O autor (2018).

A média de idade dos profissionais foi de 30,4 anos, sendo que 5 possuem idade superior e 7 possuem idade inferior à média. Dentre eles, 11 são do sexo masculino e apenas 1 do sexo feminino.

A formação dos sujeitos também foi investigada para seleção da amostra, sendo apresentada no Gráfico 4.

GRÁFICO 4: PROFISSIONAIS POR FORMAÇÃO

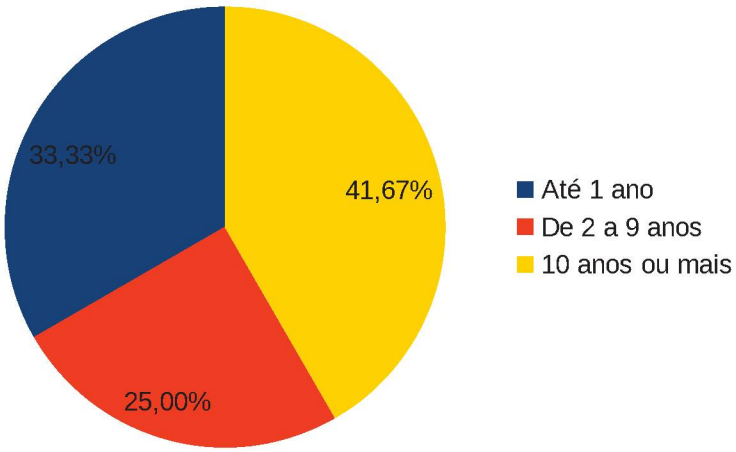


FONTE: O autor (2018).

Dentre os 12 participantes, 5 declararam que sua maior formação na área de T.I. é em nível de mestrado, 1 em nível de especialização e 6 de graduação, sendo que 1 dos graduados declarou possuir mestrado em andamento.

Outro ponto investigado diz respeito à experiência de cada profissional em desenvolvimento de *softwares*, o resultado é exibido no Gráfico 5.

GRÁFICO 5: PROFISSIONAIS POR ANOS DE EXPERIÊNCIA.



FONTE: O autor (2018).

Dos 12 participantes, 5 declararam possuir 10 ou mais anos de experiência em desenvolvimento de *softwares*, 3 declararam possuir de 2 a 9 anos e 4 menos de 1 ano.

5.2.6 Resultados das questões técnicas

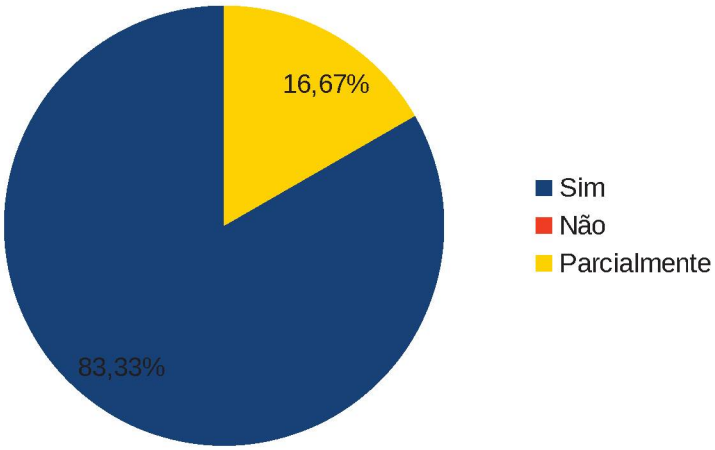
As respostas informadas nas questões técnicas detalhadas na Seção 5.2.1 são apresentadas e comentadas nos tópicos seguintes.

5.2.6.1 Resultado da questão 1.1.

Pergunta: As histórias de usuário fornecidas representam possíveis requisitos de *software*?

Dez participantes consideraram que todas as histórias de usuário do conjunto fornecido representam requisitos, respondendo “sim” à questão. Ao passo que 2 responderam “parcialmente” indicando que, no seu entendimento, pelo menos uma história do conjunto não representa um requisito de *software*, situação exibida em valores percentuais no Gráfico 6.

GRÁFICO 6: RESULTADO DA QUESTÃO 1.1.



FONTE: O autor (2018).

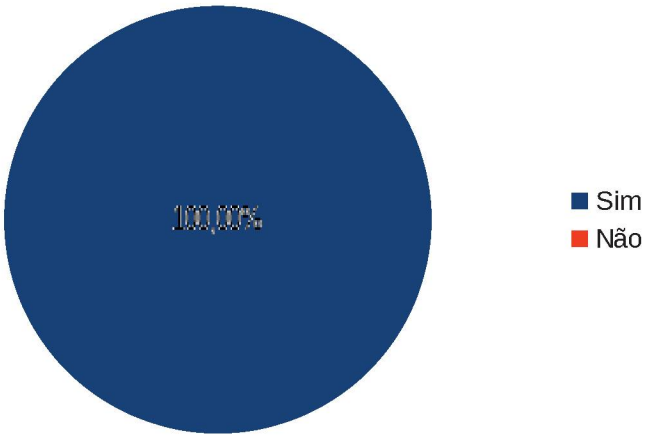
O resultado valida o conjunto de entrada submetido. Assim, pode-se entender que os textos processados pelos algoritmos de PLN do método de fato estão manipulando sentenças que, para a maioria dos desenvolvedores, especificam requisitos de *software* no formato de histórias de usuário.

5.2.6.2 Resultado da questão 1.2.

Pergunta: O conjunto desses requisitos forma um sistema ou parte de um sistema?

Todos os participantes julgaram que o conjunto fornecido forma um sistema respondendo “sim” à questão, conforme mostra o Gráfico 7.

GRÁFICO 7: RESULTADO DA QUESTÃO 1.2.



FONTE: O autor (2018).

O consenso obtido indica que, ao fazerem parte de um mesmo sistema, é alta a probabilidade de haver conexões entre as histórias de usuário. Essas

conexões são a base para a identificação de dependências que auxiliam o algoritmo de priorização a definir seu *ranking* de prioridades.

5.2.6.3 Resultado da questão 1.3.

Pergunta: Na sua opinião, quais termos representam classes candidatas no conjunto de requisitos do problema? Informe separando-as com ",".

Por se tratar de questão aberta, os dados foram agrupados em uma tabela para melhor visualização. A Tabela 9 apresenta as 19 classes candidatas informadas e a respectiva quantidade de participantes que as citou em ordem decrescente por número de citações.

TABELA 9: CLASSES CANDIDATAS INFORMADAS PELOS PROFISSIONAIS.

Classe Candidata	Citações
order	9
bill	5
stock	5
table bill	5
table	4
customer	3
drink	3
drink order	3
food	3
food order	3
kitchen	3
bar	2
menu	2
menu item	2
product	2
waiter	2
manager	1
payment	1
stock item	1

FONTE: O autor (2018).

Para comparar com a relação de classes reconhecida pelo método, foram utilizadas métricas para modelos de classificação de acordo com Leal (2017), buscando:

- O número de verdadeiros positivos (TP): casos em que uma classe candidata reconhecida pelo método consta na lista dos profissionais;

- O número de falsos positivos (FP): casos em que o método reconheceu uma classe candidata que não consta na lista dos profissionais;
- O número de falsos negativos[9] (FN): casos em que o método não reconheceu uma classe candidata que consta na lista dos profissionais;

Após a obtenção dos referidos valores, foram calculadas as seguintes métricas de classificação:

- Precisão (P): a métrica precisão objetiva responder à pergunta "Dentre os itens classificados como classes candidatas, quantos efetivamente eram?". É calculada a partir da fórmula:

  $$P = TP / (TP + FP)$$

- *Recall* (R): Métrica que tem por objetivo responder a pergunta "quando o item avaliado é realmente uma classe candidata, com qual frequência foi classificado assim?". Obtido a partir da fórmula:

  $$R = TP / (TP + FN)$$

- F1 *Score* (F1): Visa combinar precisão e *recall* de modo a gerar um número único que indique a qualidade geral do método, tendo 1 como valor ótimo. Obtido pela fórmula:

  $$F1 = 2 * P * R / (P + R)$$

Os valores e métricas citados são calculados na apresentação de resultados de cada uma das versões do Dependency Rank, compondo as Seções 5.2.7.1 e 5.2.8.1 deste trabalho.

5.2.6.4 Questão 1.4.

Pergunta: Com base nos requisitos fornecidos, qual seu *ranking* de prioridades? Informe em ordem decrescente utilizando o número das histórias de usuário.

As respostas informadas pelos participantes populam a Tabela 10. Os elementos foram posicionados respeitando a ordem de priorização informada, da esquerda para a direita.

9 O número de verdadeiros negativos não foi investigado, pois os dados informados pelos profissionais não permitem interpretar com exatidão, quais seriam os termos constantes nas histórias de usuário que deveriam ser contabilizados para essa análise, impossibilitando tal checagem.

TABELA 10: JULGAMENTO DE PRIORIDADES DOS PROFISSIONAIS.

Prof.	1º	2º	3º	4º	5º	6º	7º	8º	9º	10º
P1	R10	R9	R3	R4	R2	R1	R5	R6	R8	R7
P2	R1	R2	R3	R8	R4	R5	R6	R7	R9	R10
P3	R10	R9	R1	R2	R3	R7	R4	R5	R6	R8
P4	R10	R3	R1	R2	R7	R4	R5	R6	R8	R9
P5	R10	R9	R7	R3	R2	R1	R4	R5	R6	R8
P6	R7	R3	R1	R2	R5	R6	R4	R8	R10	R9
P7	R10	R7	R3	R1	R2	R4	R5	R9	R6	R8
P8	R3	R10	R1	R2	R6	R4	R5	R8	R9	R7
P9	R7	R3	R1	R2	R6	R9	R10	R8	R5	R4
P10	R3	R4	R1	R5	R2	R6	R7	R8	R9	R10
P11	R10	R7	R3	R1	R2	R9	R4	R5	R6	R8
P12	R7	R3	R1	R2	R4	R9	R5	R6	R8	R10

FONTE: O autor (2018).

Conforme citado anteriormente, para possibilitar a comparação com os resultados apresentados nas Seções 5.2.2 e 5.2.3 (que apresentam os resultados de saída das duas versões do protótipo), o valor alvo para o ranqueamento dos elementos foi extraído respeitando a mediana das avaliações dos profissionais.

A Tabela 11 exibe a ordenação dos requisitos nesse contexto.

TABELA 11: ORDEM DE PRIORIDADES SEGUNDO OS PROFISSIONAIS.

Requisito	Mediana	*Ranking*
R10	1,5	1
R1	3	2
R3	3	2
R2	4	4
R7	4	4
R4	6	6
R5	7	7
R9	7	7
R6	8	9
R8	9	10

FONTE: O autor (2018).

5.2.7 Discussão dos resultados do experimento com a versão 1.0

Fundamentado nos resultados extraídos após processamento dos elementos de entrada do problema pelo método, a proximidade foi avaliada considerando os dois objetivos do experimento:

a) Comparar o conjunto de classes candidatas reconhecidas pelo Dependency Rank com as classes candidatas citadas pelos profissionais;
b) Comparar o *ranking* de priorização sugerido pelo Dependency Rank com o extraído pelas avaliações dos profissionais.

As próximas seções têm por objetivo apresentar um panorama dos resultados do protótipo que implementa o método nos aspectos citados e discutir os principais fatores de influência identificados.

5.2.7.1 Identificação de classes candidatas da versão 1.0

A identificação de classes candidatas embasa a detecção de dependências do método. A exatidão alcançada nesse processo gera grande impacto na análise de prioridades implementada.

Ao julgar pelo contexto, pode-se perceber que, em geral, os termos identificados como classes candidatas podem, de fato, representar possíveis objetos de um sistema de *software* para um restaurante, com exceção do termo *task* (tarefa), que é altamente genérico.

Analisando as sentenças fornecidas, o termo que representa o papel da história de usuário (*role*) foi precisamente reconhecido em todos os casos apresentados, gerando as classes candidatas *Waiter* e *Manager*.

Com relação aos termos extraídos do objetivo das histórias de usuário (campo *goal*), pode-se perceber que a identificação também foi realizada com sucesso. Das 14 classes reconhecidas no experimento, 6 foram extraídas nesse intervalo, sendo que todas constavam na lista fornecida pelos profissionais (*Food_order*, *Drink_order*, *Table_bill*, *Order*, *Table* e *Stock*).

Coincidentemente, o campo *benefit* também identificou 6 classes candidatas (*Kitchen*, *Bar*, *Bill*, *Customer*, *Task* e *Menu*). Nesse campo foi encontrado o único falso positivo, o termo *task*. Os valores atingidos por essa versão nas métricas para modelos de classificação, apresentados na Seção 5.2.6.3 são listados a seguir:

- **Verdadeiros positivos (*true positives*): TP = 13.** Das 14 classes reconhecidas na versão, 13 apareceram na lista dos profissionais;
- **Falsos positivos (*false positives*): FP = 1.** A classe candidata *Task* foi a única não citada pelos participantes do experimento;
- **Falsos negativos (*false negatives*): FN = 6.** Dentre 19 classes candidatas informadas pelos participantes, 6 não foram reconhecidas pelo protótipo.
- **Precisão: P = 0,929**. Obtido pelo cálculo: $P=13/(13+1)$
- ***Recall*: R = 0,684.** Obtido pelo cálculo: $R=13/(13+6)$
- **F1 *Score*: F1 = 0,788.** Obtido pelo cálculo:
  $F1=2*0.929*0.684/(0.929+0.684)$

Entre os falsos negativos do experimento, os termos *Drink* e *Food*, citados por 3 profissionais, foram reconhecidos pelos algoritmos, porém, não compuseram a relação de classes gerada pois foram concatenados com a palavra *order* formando as classes *Drink_order* e *Food_order*. Tal procedimento foi implementado para possibilitar a identificação de substantivos compostos, sendo alterado após esse experimento para nova avaliação conforme Seção 4.5.1.

Duas classes candidatas identificadas pelos participantes não aparecem na relação fornecida: *Product* e *Payment*. Possivelmente foram contempladas por influência do conhecimento empírico dos avaliadores no âmbito do problema apresentado, já que o conjunto de histórias de usuário não possuía essas *strings*.

Menu_item e *Stock_item* não foram reconhecidos pelos algoritmos do Dependency Rank por não aparecerem como lemas vizinhos nos *arrays* gerados pelos métodos de PLN. A estratégia de concatenação não contemplou outro tipo de associação entre substantivos, portanto, a falta desses termos na relação final evidencia uma potencial limitação do método. Mesmo assim, pode-se perceber que dos 6 itens considerados falsos negativos, 4 (*Product*, *Payment*, *Stock_item* e *Menu_item*) necessitavam de maior carga de subjetividade para sua identificação.

A métrica de precisão obtida se destaca por ter atingido o valor 0,929, próximo ao ótimo, impulsionada pelo reconhecimento de 13 entre 14 classes candidatas pelo Dependency Rank. O valor calculado pela métrica *recall* foi 0,684, influenciado pelo número de falsos negativos do modelo (6). A combinação das referidas métricas resultou em uma avaliação geral do modelo (F1 score) de 0,788.

5.2.7.2 Priorização de requisitos da versão 1.0

O *ranking* extraído das avaliações dos profissionais é comparado com o obtido após execução do protótipo na Tabela 12. A coluna "*Ranking* automatizado" exibe o grupo em que o requisito foi classificado pelo Dependency Rank na versão 1.0, "*Ranking* profissionais" exibe o grupo em que o requisito foi classificado considerando as avaliações dos profissionais participantes no experimento. A coluna "Variação mínima" compara a posição em que o item foi classificado pelo método, tendo como valor de referência a posição no *ranking* dos profissionais, indicando a menor variação possível.

TABELA 12: VERSÃO 1.0 VS. AVALIAÇÕES DOS PROFISSIONAIS.

Requisito	***Ranking* automatizado**	***Ranking* profissionais**	**Variação mínima**
R1	Entre 9 e 10	Entre 2 e 3	6
R2	Entre 7 e 8	Entre 4 e 5	2
R3	1	2	1
R4	Entre 3 e 5	6	1
R5	2	Entre 7 e 8	5
R6	Entre 3 e 5	9	4
R7	Entre 7 e 8	Entre 4 e 5	2
R8	6	10	4
R9	Entre 3 e 5	Entre 7 e 8	2
R10	Entre 9 e 10	1	8

FONTE: O autor (2018).

Os requisitos que mais se aproximaram ao valor alvo de classificação foram R3 e R4, que atingiram variação mínima de 1 posição. Levando em conta o tamanho e a segmentação do conjunto, as classificações de R2, R7 e R9, com 2 posições de variação mínima representam distância do valor alvo de 20% a 40% nos casos de R2 e R7 e de 20% a 50% no caso de R9.

Sob esse ponto de vista, os demais elementos, R1, R5, R6, R8 e R10 não foram classificados com proximidade razoável. As razões para as diferenças de classificação entre os *rankings* são variadas. Para entendê-las é preciso enfatizar que o único critério utilizado pelo Dependency Rank é quantidade de dependências encontradas automaticamente, apoiado por métodos de PLN. Já a avaliação dos profissionais contou com critérios pessoais e subjetivos, muitas vezes não implementáveis computacionalmente.

A variação desses critérios pode ser melhor observada na Tabela 10 da Seção 5.2.6.4, que mapeia o julgamento de prioridades de cada profissional. Pegando como exemplo R7, vemos que 5 profissionais o julgaram como sendo de prioridade muito alta (1ª ou 2ª posição), 2 como média/alta (entre 3ª e 5ª), 3 como média/baixa (entre 6ª e 8ª) e outros 2 como prioridade baixa (9ª ou 10ª posições), o que evidencia a diferença nas análises pessoais.

Sobre os itens julgados como prioritários pelos profissionais, percebemos que R3 e R10 são classificados com alta prioridade por serem básicos para outras funcionalidades do sistema proposto. R3 descreve a necessidade de abrir a conta da mesa para que seja possível adicionar pedidos, enquanto R10 descreve a necessidade de gerenciar o estoque para habilitar itens do menu. Portanto, o entendimento de que sem a implementação de R3 e R10 outros requisitos como R1, R2, R4, e *etc*. não seriam relevantes deve ter elevado as mencionadas prioridades.

Entre os requisitos de prioridade média/alta, a presença de R7 se deve, provavelmente, ao mesmo aspecto que priorizou R3 e R10, já que descreve precedência para abrir uma conta vazia. R1 e R2 receberam prioridade alta pois contemplam as funcionalidades principais do sistema, isto é, criar um pedido para a cozinha ou para o bar e encontram-se, logicamente, depois de R3, seu precedente.

Já R4, R5 e R6, que receberam prioridade média/baixa dos profissionais, têm como característica em comum serem dependentes dos citados nos grupos 1 e 2. R4 e R5 por motivos idênticos, já que descrevem a alteração ou cancelamento de um pedido, que não podem ser feitos sem que ele seja criado (R1/R2). R6 descreve o fechamento de uma conta que, para tanto, precisa ter sido aberta (R3).

Os itens menos prioritários se referem à impressão de uma conta (R8) e à visualização de uma lista de pedidos. Provavelmente sua classificação foi influenciada pelo subjetivo nível de importância de suas implementações para o problema, pois não caracterizam uma funcionalidade básica.

Sendo assim, pode-se notar que a identificação de alguns fatores que influenciaram as decisões de priorização não foram contemplados na implementação do método. Alguns deles, como a precedência entre requisitos podem ser adicionados realizando uma análise dos verbos principais envolvidos nas histórias de usuário. Tal implementação possibilitaria, por exemplo, classificar *create*,

reconhecido em R1 e R2, com prevalência a *change* e *cancel*, melhorando a qualidade do ranqueamento automatizado.

5.2.8 Discussão dos resultados do experimento com a versão 1.1

A presente seção confronta os resultados atingidos pela versão 1.1 do Dependency Rank, apresentados na Seção 5.2.3, com a análise dos profissionais, focando nas respostas submetidas às questões 1.3 e 1.4 do questionário, detalhadas nas seções 5.2.6.3 e 5.2.6.4, respectivamente.

5.2.8.1 Identificação de classes candidatas da versão 1.1

Assim como ocorreu com a versão 1.0, a nova implementação identificou corretamente a grande maioria das classes candidatas citadas pelos profissionais. A principal diferença notada diz respeito ao reconhecimento dos termos *Food* e *Drink* como classes do problema, estes termos foram ignorados pela versão anterior pela exigência de concatenação dos substantivos relacionados, situação corrigida para a versão 1.1.

Para o problema submetido, a proximidade da versão 1.1 foi superior ao julgar pelas métricas para modelos de classificação adotadas. Os valores atingidos, apresentados na Seção 5.2.6.3 são listados a seguir:

- **Verdadeiros positivos (*true positives*): TP = 15.** A versão 1.1 encontrou 16 classes candidatas, das quais 15 constam na lista dos profissionais.
- **Falsos positivos (*false positives*): FP = 1.** A classe candidata *Task* foi a única não citada pelos participantes do experimento;
- **Falsos negativos (*false negatives*): FN = 4.** Nessa versão, apenas 4 classes não foram reconhecidas pelo método.
- **Precisão: P = 0,938**. Obtido pelo cálculo: $P=15/(15+1)$
- ***Recall*: R = 0,789.** Obtido pelo cálculo: $R=15/(15+4)$
- **F1 *Score*: F1 = 0,857.** Obtido pelo cálculo:

  $F1=2*0.938*0.789/(0.938+0.789)$

O reconhecimento dos termos *Food* e *Drink* possibilitou elevar o valor de TP para 15, sendo que anteriormente havia ficado em 13. Não houve alteração na quantidade de FP, que se manteve em 1 pela inclusão do termo *Task*, não citado pelos profissionais. Enquanto isso, FN teve seu valor alterado para 4, já que 2

termos anteriormente classificados como FN passaram a ser classificados como TP nessa versão.

Outras alterações podem ser constatadas nas métricas de precisão, *recall* e F1 *score*. A maior alteração foi percebida em R, que obteve 0,789, ante 0,684 na versão 1.0, o que significa uma melhoria no valor de *recall* superior a 15%. A métrica P teve seu valor alterado de 0,929 para 0,938, representando melhoria próxima a 1%. Já a qualidade geral do modelo, calculada por F1, foi melhorada em pouco mais de 8%, atingindo 0,857 ante 0,788 na versão 1.0.

5.2.8.2 Priorização de requisitos da versão 1.1

A Tabela 13 compara o *ranking* extraído das avaliações dos profissionais com o obtido pela versão 1.1 do método. Assim como na Tabela 12 da Seção 5.2.7.2, a coluna "*Ranking* automatizado" exibe o grupo em que o requisito foi classificado pelo Dependency Rank na versão 1.0, "*Ranking* profissionais" exibe o grupo em que o requisito foi classificado considerando as avaliações dos profissionais participantes no experimento. A coluna "Variação mínima" compara a posição em que o item foi classificado pelo método, tendo como valor de referência a posição no *ranking* dos profissionais, indicando a menor variação possível.

TABELA 13: VERSÃO 1.1 VS. AVALIAÇÕES DOS PROFISSIONAIS.

Requisito	***Ranking* automatizado**	***Ranking* profissionais**	**Variação mínima**
R1	Entre 2 e 6	Entre 2 e 3	0
R2	Entre 2 e 6	Entre 4 e 5	0
R3	1	2	1
R4	Entre 2 e 6	6	0
R5	Entre 2 e 6	Entre 7 e 8	1
R6	Entre 7 e 9	9	0
R7	Entre 7 e 9	Entre 4 e 5	2
R8	Entre 7 e 9	10	1
R9	Entre 2 e 6	Entre 7 e 8	1
R10	10	1	9

FONTE: O autor (2018).

Observando a Tabela 13, percebe-se que 4 dos 10 requisitos, quais sejam, R1, R2, R4 e R6, foram classificados virtualmente na posição correta considerando como valor alvo o *ranking* extraído das avaliações dos profissionais.

Para 4 requisitos a classificação do Dependency Rank ficou com variação mínima de 1 posição, podendo ser considerado satisfatório, sendo, R3, R5, R8 e R9. Os requisitos que mais se afastaram das posições alvo nesse cenário foram R7, com 2 posições de variação mínima e R10, com 9. O caso específico de R10 pode ser melhor entendido ao analisar a Tabela 10 da Seção 5.2.6.4, onde mesmo sendo considerado prioritário pelo critério da mediana das avaliações, 4 profissionais o classificaram como de baixa prioridade, atribuindo as duas últimas posições do *ranking* ao referido requisito, estando de acordo com a análise do método.

É importante enfatizar para esses casos que, as várias situações de empate constatadas em especial entre a 2ª e 6ª posição contribuem para uma adequação mais fácil dos requisitos no *ranking*, que não esteve tão segmentado quanto o gerado pela versão anterior.

A diferença predominante em relação à versão 1.0 é a quantidade de dependências identificada em R3, que passou a ter 8, não possuindo *links* apenas para R10. Isso se deve à interpretação do termo *Order* como classe candidata em outros requisitos do conjunto. O entendimento parece coerente, já que dos 12 participantes, 10 indicaram que R3 estaria entre os 3 requisitos de maior prioridade para esse problema, sendo que para 2 participantes, ocuparia exatamente a primeira posição.

Além de R3, o julgamento do método esteve em conformidade com, no mínimo, a metade dos participantes em mais 6 casos:

- R1 (entre 2 e 6): recebeu 11 das 12 avaliações no intervalo;
- R2 (entre 2 e 6): recebeu todas as avaliações no intervalo;
- R4 (entre 2 e 6): recebeu 7 das 12 avaliações no intervalo;
- R6 (entre 7 e 9): recebeu 8 das 12 avaliações no intervalo;
- R8 (entre 7 e 9): recebeu 7 das 12 avaliações no intervalo;
- R9 (entre 2 e 6): recebeu 6 das 12 avaliações no intervalo;

Nesse sentido, R5 e R7 foram o que mais se afastaram dos valores alvo. R5 obteve apenas 3 avaliações no intervalo entre a 2ª e 6ª posições, enquanto R7 obteve apenas 2 no seu intervalo, entre a 7ª e 9ª posições.

5.2.9 Comparativo entre os *rankings* das versões no experimento

Após o processamento da entrada pelas duas versões do Dependency Rank, as quantidades de dependência detectadas em cada requisito e suas respectivas posições nos *rankings* são exibidos na Tabela 14.

TABELA 14: VERSÃO 1.0 VS. VERSÃO 1.1.

Requisito	Deps. V. 1.0	Deps. V. 1.1	*Ranking* V. 1.0	*Ranking* V. 1.1
R1	0	5	9	2
R2	1	5	7	2
R3	5	8	1	1
R4	3	5	3	2
R5	4	5	2	2
R6	3	3	3	7
R7	1	3	7	7
R8	2	3	6	7
R9	3	5	3	2
R10	0	0	9	10

FONTE: O autor (2018).

Nota-se que a nova implementação da busca de classes candidatas teve impacto significativo principalmente na priorização dos requisitos R1, R2 e R6. Para R1 e R2, que tiveram seus números de dependência aumentados pela nova implementação em razão do reconhecimento da classe candidata *Order*, o valor de priorização atribuído na versão 1.1 do método foi bastante superior ao anterior, visto que R1 ocupava a 9ª e R2 a 7ª posição, e passaram à 2ª posição na versão 1.1.

O número de dependências foi alterado de 0 para 5 em R1, de 1 para 5 em R2, de 5 para 8 em R3, de 3 para 5 em R4, de 4 para 5 em R5, de 1 para 3 em R7, de 2 para 3 em R8 e de 3 para 5 em R9. Enquanto isso, R6 e R10 foram os únicos a não terem mais dependências reconhecidas na nova versão, nesses casos a posição no *ranking* apresentou a maior queda.

5.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou os experimentos realizados para validar o método desenvolvido nessa pesquisa. Os dois cenários em que o Dependency Rank foi testado foram detalhados exibindo conjuntos de entrada e resultados alcançados pelas duas versões implementadas, comparando os resultados das versões a fim de identificar qual delas obteve maior proximidade.

No experimento que avaliou um sistema real já priorizado, o *ranking* fornecido pela equipe de desenvolvimento foi comparado ao gerado após execução do protótipo. A versão 1.0 classificou corretamente 60% dos requisitos do conjunto nos cenários de alta e baixa prioridade. Contudo, constatou-se que a versão 1.1 atingiu valores mais próximos ao desejado já que, mesmo com grupos de prioridade mais heterogêneos, houve maior quantidade de acertos, sendo 3 contra 1. Além disso, outros 5 requisitos receberam variações mínimas de até 3 posições, ou seja, considerando o conjunto de 14 requisitos e o valor mínimo, essa variação foi inferior a 22%.

O segundo experimento, que contou com a colaboração de profissionais avaliando um sistema ainda não priorizado, confrontou tanto os *rankings* sugeridos pelos profissionais com o obtido após execução do protótipo, quanto as classes candidatas envolvidas. Em ambos os casos a proximidade da versão 1.1 foi superior, melhorando a métrica de precisão em aproximadamente 1%, a de *recall* em mais de 15% e a F1 *score* em pouco mais de 8%, fato que reforça a tese de melhora com a nova implementação que considerou, para os casos de classes candidatas compostas por mais de um substantivo, todos os termos como classes candidatas do problema.

O *ranking* de prioridades da versão 1.1 também ficou mais próximo ao valor alvo considerando os grupos gerados após a execução dos algoritmos do método. Em 7 casos (70%), o intervalo de posições sugerido pelo método esteve em conformidade com, no mínimo, a metade dos profissionais. Na versão anterior, com mais grupos, portanto, de menor probabilidade de acerto, o número de requisitos classificados no mesmo intervalo conforme julgamento de parte dos avaliadores foi 5 (50%).

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho teve como objetivo desenvolver um método de PRS, denominado Dependency Rank, capaz de identificar automaticamente as dependências entre o conjunto de requisitos de um *software* utilizando ferramentas de PLN. Para isso, foram abordados os conceitos de requisitos de *software*, desenvolvimento ágil, histórias de usuário, interdependências entre requisitos, PRS, PLN, preprocessamento de texto, POS *tagging* e lematização.

Como suporte, foram investigados artigos publicados na última década na área de PRS em um estudo sobre o estado da arte, realizado por mapeamento sistemático de literatura. Após o levantamento, percebeu-se a falta de pesquisas que ambicionem a detecção automatizada dessas dependências na área, fato que torna as avaliações excessivamente dependentes de trabalho humano.

Um método de priorização por *ranking* capaz de identificar as dependências com auxílio de ferramentas de PLN foi desenvolvido, gerando um protótipo implementado em linguagem de programação Java e o *framework* OpenNLP, sendo avaliado em dois experimentos. Em um deles, um conjunto de histórias de usuário de um problema real, já priorizado, foi submetido como entrada para o método, que gerou seu *ranking* para posterior comparação. No outro, um conjunto de histórias de usuário de um sistema ainda não priorizado foi utilizado como entrada. Nesse caso, foram comparadas as listas de classes candidatas e prioridades obtidas pelo método com as dos profissionais.

Como resultados, o método obteve, em sua versão de maior proximidade, a 1.1, precisão de 0,938, *recall* de 0,789 e F1 *score* de 0,857 na identificação de classes candidatas, que direcionam a detecção de dependências entre os requisitos nos dois cenários explorados.

O *ranking* de priorização sugerido alcançou valores próximos ao julgamento humano nos dois cenários. No primeiro experimento, 3 requisitos foram classificados corretamente considerando o valor alvo e a outros 5 foram atribuídas posições com variação mínima inferior a 22%. No segundo experimento, o *ranking* sugerido classificou 70% dos requisitos do conjunto no mesmo intervalo que, pelo menos, a metado dos julgamentos dos profissionais participantes.

As principais limitações identificadas nessa pesquisa foram:

a) Abordagem orientada a substantivos: os verbos principais, ainda que sejam identificados, não compõem parâmetros de avaliação para priorização dos requisitos;
b) Associação de substantivos: na atual implementação, não é possível associar substantivos que não estejam em endereços vizinhos no *array* de lemas;
c) Carga de subjetividade: a implementação atual não contempla estratégias de aprendizagem de máquina sobre o contexto de aplicação do sistema em análise;
d) Precedência entre requisitos: ao constatar uma dependência, não foram implementados métodos auxiliares para identificação de precedência entre os requisitos;
e) Realização de testes em problemas maiores: os dois cenários explorados nos experimentos descrevem problemas pequenos, a aplicação em problemas maiores e mais complexos pode evidenciar outros pontos a investigar.

Diante disso, identifica-se como possibilidades de trabalhos futuros, a implementação de rotinas que tenham como objetivo resolver ou, pelo menos, amenizar as limitações citadas nos itens a, b, c e d.

Finalmente, conclui-se que a adoção do critério de priorização de requisitos baseado em dependências identificadas por ferramentas de PLN, pode auxiliar desenvolvedores na tarefa de priorização. Ainda que não forneça resultados exatos, os experimentos mostram que é possível alcançar resultados aproximados.

Sendo assim, a utilização do método proposto em processos de *software* que contem com um grande conjunto de requisitos diminuiria a quantidade de tempo/esforço necessário para o julgamento dessas prioridades ou definição de incrementos, pois fornece uma perspectiva sobre as dependências entre eles.

Complementarmente, a identificação do verbo principal, que direciona a ação do requisito, auxilia na análise de precedência, podendo ser um critério auxiliar nessa tarefa.

REFERÊNCIAS

ACHIMUGU, Philip; SELAMAT, A.; IBRAHIM, R.; MAHRIN, M. N. R. A systematic literature review of software requirements prioritization research. **Information and software technology**, v. 56, n. 6, p. 568-585. 2014.

ALAWNEH, Luay. Requirements prioritization using hierarchical dependencies. **Information Technology-New Generations**, Springer, Cham, p. 459-464. 2018.

ALZYOUDI, Ruba; ALMAKADMEH, Khaled; NATOUREAH, Hutaf. A Probability Algorithm for Requirement Selection In Component-Based Software Development. **Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication**, ACM, p. 95. 2015.

APPOLINÁRIO, Fábio. **Metodologia da ciência:** filosofia e prática da pesquisa. São Paulo: Cengage Learning, 2012. ISBN 978-85-221-1177-0.

APACHE SOFTWARE FOUNDATION, THE. **OpenNLP**, 2017. Disponível em: <opennlp.apache.org>. Acesso em: 29 ago. 2018.

BAPTISTA, M. N.; CAMPOS, D. C. **Metodologia de pesquisa em ciências:** análises quantitativa e qualitativa. Rio de Janeiro: LTC, 2007.

BIRD, Steven; KLEIN, Ewan; LOPER, Edward. **Natural language processing with Python:** analyzing text with the natural language toolkit. Sebastopol, CA: O'Reilly Media, Inc., 2009.

BOURQUE, Pierre; FAIRLEY, R. E. **Guide to the software engineering body of knowledge (SWEBOK (R)).** IEEE Computer Society Press, 2014.

BRERETON, Pearl; KITCHENHAM, B. A.; BUDGEN, D.; TURNER, M.; KHALIL, M. Lessons from applying the systematic literature review process within the software engineering domain. **Journal of systems and software**, v. 80, n. 4, p. 571-583. 2007.

CAMBRIA, Erik; WHITE, Bebo. Jumping NLP curves: A review of natural language processing research. **IEEE Computational intelligence magazine**, v. 9, n. 2, p. 48-57. 2014.

CARLSHAMRE, Pär; SANDAHL, K.; LINDVALL, M.; REGNELL, B.; OCH DAG, J. N. An industrial survey of requirements interdependencies in software product release planning. **Proceedings Fifth IEEE International Symposium on Requirements Engineering.** IEEE, p. 84-91. 2001.

CONDORI-FERNANDEZ, Nelly; GRANDA, Maria Fernanda; VOS, Tanja EJ. Towards a functional requirements prioritization with early mutation testing. **Proceedings of the 5th International Workshop on Requirements Engineering and Testing.** ACM, p. 21-24. 2018.

CRESWELL, John W. Procedimentos de métodos mistos. In: Creswell, JW **Projeto de Pesquisa:** Métodos qualitativo, quantitativo e misto. Porto Alegre: Artmed, 2007. p. 211-227, 2007.

DA SILVA, Alberto Manuel Rodrigues; VIDEIRA, Carlos Alberto Escaleira. **UML, metodologias e ferramentas CASE.** Porto/Lisboa: Centro Atlântico, 2001.

DAHLSTEDT, Asa G.; PERSSON, Anne. Requirements interdependencies-moulding the state of research into a research agenda. **Proceedings of Ninth International Workshop on Requirements Engineering.** Klagenfurt/Velden, Austria. p. 55-64. 2003.

DESHPANDE, Dhanraj. Textual Requirement Analysis for Object Model Designing by Using NLP. **International Journal of Innovative Research in Science, Engineering and Technology**, v. 1, n. 2. 2012.

FITSILIS, Panos; GEROGIANNIS, Vassilis; ANTHOPOULOS, Leonidas; SAVVAS, Ilias K. Supporting the requirements prioritization process using social network analysis techniques. **19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises**. IEEE, p. 110-115. 2010.

GALLIERS, Julia Rose; JONES, K. Sparck. **Evaluating natural language processing systems.** Cambridge: University of Cambridge, 1993. Relatório técnico.

GILLAIN, Joseph; JURETA, Ivan; FAULKNER, Stéphane. Planning optimal agile releases via requirements optimization. **2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)**. IEEE, p. 10-16. 2016.

INDURKHYA, Nitin; DAMERAU, Fred J. (Ed.). **Handbook of natural language processing.** Cambridge: CRC Press, 2010.

LEAL, RS. **Métricas Comuns em Machine Learning: como analisar a qualidade de chat bots inteligentes.** 2017. Em: <medium.com/as-máquinas-que-pensam/métricas-comuns-em-machine-learning-como-analisar-a-qualidade-de-chat-bots-inteligentes-introdu-57ff30424192>. Acesso em: 14 ago. 2018.

LI, Yan; ZHANG, Man; YUE, Tao; ALI, Shaukat; ZHANG, Li. Search-based Uncertainty-wise Requirements Prioritization. **2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS).** IEEE, p. 80-89. 2017.

MALL, Rajib. **Fundamentals of software engineering.** Delhi: PHI Learning Pvt. Ltd., 2018.

Manifesto para o desenvolvimento ágil de software. 2001. Disponível em: <http://www.manifestoagil.com.br/>. Acesso em 05 set. 2018.

MISAGHIAN, Negin; MOTAMENI, Homayun. An approach for requirements prioritization based on tensor decomposition. **Requirements Engineering**, v. 23, n. 2, p. 169-188. 2018.

NOGUEIRA, E. D. A. **Lista de histórias de usuário priorizadas** [mensagem pessoal]. Mensagem recebida por <marlonbsi@gmail.com> em 22 set. 2018.

OGNJANOVIC, Ivana; GASEVIC, D.; BAGHERI, E.; ASADI, M. Conditional preferences in software stakeholders' judgments. **Proceedings of the 2011 ACM Symposium on Applied Computing.** ACM, p. 683-690. 2011.

PAULA FILHO, Wilson de Pádua. **Engenharia de software.** Rio de Janeiro: LTC, 2003.

PETERSEN, Kai; FELDT, Robert; MUJTABA, Shahid; MATTSSON, Michael. Systematic mapping studies in software engineering. **Ease**, p. 68-77. 2008.

PETERSEN, Kai; VAKKALANKA, Sairam; KUZNIARZ, Ludwik. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, v. 64, p. 1-18. 2015.

PITANGUEIRA, Antônio Mauricio; MACIEL, Rita Suzana P.; BARROS, Márcio. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. **Journal of Systems and Software**, v. 103, p. 267-280. 2015.

POHL, Klaus. PRO-ART: Enabling requirements pre-traceability. **Proceedings of the second international conference on requirements engineering.** IEEE, p. 76-84. 1996.

PRESSMAN, R. S. **Software Engineering:** A practitioner's approach. New York, NY: McGraw-Hill Education, 2010.

PRESSMAN, R. S; MAXIM, B. R. **Engenharia de Software:** Uma abordagem profissional. São Paulo: McGraw-Hill Education, 2016.

PRINCETON UNIVERSITY. **WordNet:** A Lexical Database for English. New Jersey, USA, 2018 em: <wordnet.princeton.edu>. Acesso em: 29 ago. 2018.

PITANGUEIRA, Antônio Mauricio; MACIEL, Rita Suzana P.; BARROS, Márcio. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. **Journal of Systems and Software**, v. 103, p. 267-280. 2015.

REESE, RM. **Natural Language Processing with Java.** Birmingham: Packt Publishing, 2015. ISBN: 9781784391799.

REGNELL, Björn; KUCHCINSKI, Krzysztof. Exploring software product management decision problems with constraint solving-opportunities for prioritization and release planning. **Software Product Management (IWSPM), 2011 Fifth International Workshop**, IEEE, p. 47-56. 2011.

SCHWEINBERGER, Martin. **Part-Of-Speech Tagging with R.** 2016. Disponível em: <http://martinschweinberger.de/docs/articles/PosTagR.pdf>. Acesso em: 02 out. 2018.

SHAO, Fei; PENG, Rong; LAI, Han.; WANG, Bangchao. DRank: A semi-automated requirements prioritization method based on preferences and dependencies. **Journal of Systems and Software**, v. 126, p. 141-156. 2017.

SREE-KUMAR, Anjali; PLANAS, Elena; CLARISÓ, Robert. Extracting software product line feature models from natural language specifications. **Proceeedings of the 22nd International Conference on Systems and Software Product Line**, ACM, v. 1, p. 43-53. 2018.

SUREKA, Ashish. Requirements prioritization and next-release problem under Non-additive value conditions. **Software Engineering Conference (ASWEC)**, IEEE, 23rd Australian, p. 120-123. 2014.

TONELLA, Paolo; SUSI, Angelo; PALMA, Francis. Using interactive GA for requirements prioritization. **2nd International Symposium on Search Based Software Engineering.** IEEE, p. 57-66. 2010.

TONELLA, Paolo; SUSI, Angelo; PALMA, Francis. Interactive requirements prioritization using a genetic algorithm. **Information and software technology**, v. 55, n. 1, p. 173-187. 2013.

WIEGERS, Karl. First things first: prioritizing requirements. **Software Development**, v. 7, n. 9, p. 48-53. 1999.

WOHLIN, Claes. Guidelines for snowballing in systematic literature studies and a replication in software engineering. **Proceedings of the 18th international conference on evaluation and assessment in software engineering**, ACM, p. 38. 2014.

YUE, Tao; ALI, Shaukat. Applying search algorithms for optimizing stakeholders familiarity and balancing workload in requirements assignment. **Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation**, ACM, 2014. p. 1295-1302.

ZHANG, Zhang. **Specifying Functional Requirements Dependency in the REWiki.** 53 f. Dissertação (M.Sc. Computer Science) - University of Tampere, Tampere, 2013.

APÊNDICE 1 – QUESTIONÁRIO DE AVALIAÇÃO DO MÉTODO

Avaliação do método de priorização de requisitos a partir de dependências identificadas por NLP

Responda as questões de acordo com o problema fornecido

Para responder as questões de 1 a 4 considere o conjunto de histórias de usuário abaixo:

US01: As a waiter I want to create a food order so that the kitchen can prepare it.
US02: As a waiter I want to create a drink order so that the bar can prepare it.
US03: As a waiter I want to open the table bill so that I can add orders.
US04: As a waiter I want to change an order so that I can fix a wrong order.
US05: As a waiter I want to cancel an order so that I can warn the kitchen or the bar.
US06: As a waiter I want to close the table bill so that the bill can be calculated.
US07: As a waiter I want to set the table so that I can open an empty bill.
US08: As a waiter I want to print the bill so that I can deliver it to the customer.
US09: As a manager I want to view the list of orders so that I can distribute tasks.
US10: As a manager I want to manage the stock so that I can enable items on the menu.

1. **1. As histórias de usuário fornecidas representam possíveis requisitos de software?**
   Mark only one oval.
   - ◯ Sim
   - ◯ Parcialmente
   - ◯ Não

2. **2. O conjunto desses requisitos forma um sistema ou parte de um sistema?**
   Mark only one oval.
   - ◯ Sim
   - ◯ Não

3. **3. Na sua opinião, quais termos representam classes candidatas no conjunto de requisitos do problema? Informe separando-as com ",".**

4. **4. Com base nos requisitos fornecidos, qual seu ranking de prioridades? Informe em ordem decrescente utilizando o número das histórias do usuário.**

Para finalizar complete com seu perfil

5. **Seu ano de nascimento (aaaa):**

6. **Sexo:**
   Mark only one oval.
   - ◯ Masculino
   - ◯ Feminino

7. **Formação na área de T.I. - informar nível e nome do(s) curso(s) ou "nenhum" caso não possua:**

8. **Experiência em desenvolvimento de software (em anos):**